Proceedings of the Workshop on the Wrapper Techniques for Legacy Systems

Thiran, Philippe; van den Heuvel, Willem-Jan

Publication date:
2004

Technische Universiteit Eindhoven
Department of Mathematics and Computer Science

WRAP 2004 - First International Workshop on Wrapper Techniques for Legacy Systems

In connection with the 11th Working Conference on Reverse Engineering

Workshop Proceedings

Editors:

Philippe Thiran and Willem-Jan van den Heuvel

04/34

ISSN 0926-4515

editors: prof.dr. P.M.E. De Bra
prof.dr.ir. J.J. van Wijk

Reports are available at:
http://library.tue.nl/catalog/TUEPublication.csp?Language=dut&Type=ComputerScienceReports&Sort=Author&level=1 and
http://library.tue.nl/catalog/TUEPublication.csp?Language=dut&Type=ComputerScienceReports&Sort=Year&Level=1

Computer Science Reports 04/34
Eindhoven, November 2004

Preface

Welcome to WRAP 2004, the first international workshop on Wrapper Techniques for Legacy Systems. This workshop concentrates on challenging research topics regarding the development of wrappers for legacy systems.

Research and development in this field are of paramount importance as legacy systems hold services that remain useful beyond the means of the technology in which they were originally implemented. Legacy wrappers are used to facilitate reuse of key portions of the legacy systems for their integration into novel business processes and applications, e.g., for enacting virtual supply chains. Wrappers provide a clean way for systems to interact with one another as they allow the encapsulation of legacy systems into cohesive and reusable black-box components with self-describing interfaces.

Up till now, many approaches for constructing legacy wrappers solely focus on legacy systems, whereas new business processes remain largely neglected. Also, the maintenance of wrappers has not gained much attention; this is strange, as wrappers must be designed in such a way that they allow graceful adaptation to accommodate new business requirements that may emerge over time. At the same time, new initiatives and technologies, such as OMG Model Driven Architecture, open up new possibilities for constructing wrappers.

WRAP 2004 is intended to bring together researchers and practitioners interested in various issues regarding the design, development and maintenance of wrappers for legacy systems in the context of modern business processes. In particular, it aims at assessing and exploring current and novel techniques and methodologies for wrapping legacy systems, and identifying important future research directions.

We received a substantial number of submissions from ten countries. After a review process including reviews and recommendations by at least two members of the program committee for each paper, four papers were selected for publication.

We wish to thank the program committee and all the authors who submitted papers for the workshop. We would like to thank the Department of Mathematics and Computer Science at the Technische Universiteit Eindhoven, The Netherlands, for its support of the Workshop. Finally, we would like to thank the WCRE 2004 organizing committee for their effort and support in making this workshop possible.

Philippe Thiran and Willem-Jan van den Heuvel
WRAP 2004 Workshop Chairs

Workshop Organizers

Philippe Thiran, *Eindhoven University of Technology*, The Netherlands
Willem-Jan van den Heuvel, *University of Tilburg*, The Netherlands

Program Committee

Djamal Benslimane, *University of Lyon I*, France
Henning Christiansen, *Roskilde University*, Denmark
Vincent Englebert, *University of Namur*, Belgium
Chirine Ghedira, *University of Lyon I*, France
Jean Henrard, *ReveR - IS Reengineering*, Belgium
Geert-Jan Houben, *Eindhoven University of Technology*, The Netherlands
Zakaria Maamar, *Zayed University*, United Arab Emirates
Heiner Stuckenschmidt, *Free University of Amsterdam*, The Netherlands
Philippe Thiran, *Eindhoven University of Technology*, The Netherlands
Willem-Jan van den Heuvel, *University of Tilburg*, The Netherlands

Contents

Heterogeneous Data Extraction in XML

Jorge Vila[1] and Carmen Costilla[2]

[1] Researcher at the *Information Systems and Databases* (SINBAD) Research Group
jvila@sinbad.dit.upm.es, http://sinbad.dit.upm.es

[2] Titular Professor, Main Researcher of SINBAD Research Group
Dept. Ingeniería de Sistemas Telemáticos (DIT), Technical University of Madrid (UPM),
costilla@dit.upm.es, http://www.dit.upm.es,
http://sinbad.dit.upm.es

Abstract. A huge amount of heterogeneous information is daily published on the web, mainly based on three technologies: static HTML web pages, dynamic SQL/ XML web sites and, recently, the OWL semantic web. The data extraction is the task of recognizing and extracting specific data fragments from a collection of documents presented as a query result. This work proposes a **Data Extractor Model** inside a virtual and dynamic web integrated architecture for multiple heterogeneous Digital Archives data sources. Belonging to this model, we describe in depth the **Data Extraction i**, a main component in charge of hiding any particular syntactic heterogeneity (format and content structure) coming from any digital archive. In fact, the **Data Extraction i** deals with HTML web pages and databases (O-RDB and XML-DB) web sites. Through a XML Descriptor for each Digital Archive, it is specified how to invoke particular local queries. In this way, the **Data Extraction i** component provides a first uniform layer to the upper ones. The paper discusses some key issues involved in extracting and translating heterogeneous data (the bottom abstraction level) into a syntactically homogeneous XML information and models each **Data Extraction i**.

1 Introduction

A huge amount of heterogeneous information is daily published on the web. Its content is mainly based on three kinds of technologies: a) static HTML web pages, at the beginning; b) dynamic web sites -generated from databases and XML- with increasing popularity; and c) the semantic web -RDF/DAML+ OIL/OWL coded- a current important research topic.

Digital Archive (here after, **DA**) is a valuable application within the Web Information Systems for accessing digitized documentary data. They belong to the documentary world (libraries, archives and museums) and contain a paramount documentary information about human activity (cultural heritage, institutions, business, etc).

Unfortunately, most web DA are offered as static HTML handmade pages. In [8], we have described valuable web DA proposals: the OAI (Open Archive Initiative)

[25] and the Fedora [15] projects. Both specify remote web access protocols to a concise DA.

In this context, our research [6,7,8] is focused on a virtual and dynamic web integrated architecture related to multiple heterogeneous DA data sources, as figure 1 represents. We are investigating a generic solution for the (semi-) automatic integration of multiple web DA, allowing queries against several DA independently of their location and content (as a unique DA).

Our starting point was the *Parliamentary Integrated Management System* (*SIAP*), we have built and is successfully running at the *Asamblea de Madrid* since 1999. *SIAP* contains a DA Management System whose main screen in figure 1 is called '*Data Source Asamblea de Madrid,* ***Digital Archive N***' (www.crcit.es/SIAP) [5]. Thanks to *SIAP* the *Asamblea of Madrid* has totally digitized its documentary fonds, since the beginning of our democracy, stored in an object-relational database.

Immersed in this architecture, this paper describes the ***Data Extractor Model***, the layer in charge of the information extraction from heterogeneous data sources. This layer is composed of two main components: the ***Data Extraction i*** (here after, ***DE***) related to a particular DA -that is the main topic of this paper- and the *Wrapper*. The *DE* is in charge of showing the underlying data sources in an uniform syntactic way to its respective *Wrapper,* as figure 2 represents.

Each *DE*, as an interface to the *Wrapper*, makes possible to access any participant DA in a particular integration. In this sense, we propose a data extraction tool, showing the sources in a XML generic way.

The remainder of this paper is organized as follows. Section 2 describes the web integrated architecture highlighting its *Data Extractor Model.* Section 3 deals in-depth with the *Data Extraction* component, *DE*. Section 4 describes the extraction process adapted to each type of data source. Section 5 gives an overview about how a *DE* is running. Finally, the conclusions are summarized in Section 6.

2 Architecture

Achieving heterogeneous web DA integration is an important research topic at the SINBAD-UPM group. As well as other aspects, we need to provide an architecture that allows the web user to access several heterogeneous DA transparently. The final idea is the architecture implementation through web services organized in levels, as described in [11], and compliant with J2EE platform. However, web services architecture does not solve the semantic integration problem: we think that the problem is the lack of a global ontology. We have modelled the DA in UML [26] and the definition of the virtual web DA integrated architecture is given in [8].

As figure 1 summarizes, this architecture is based on a *Mediator* layer (composed of ontologies, mappings and data repositories) and a *Wrappers* layer (saving the data sources heterogeneity using XML). Besides, figure 2 shows the ***Unified Ontological Model*** and ***Data Extractor Model*** layers, corresponding to the *Mediator* and *Wrappers* layers. The data sources remain as localities without changing their own structures.

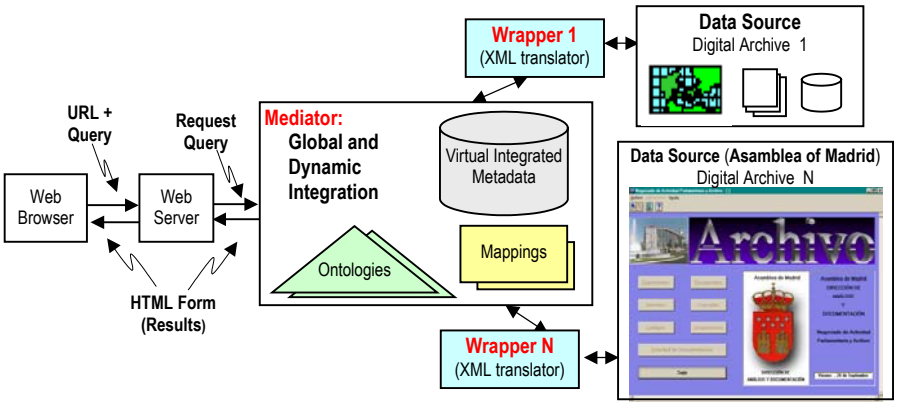

Figure 1. Web Digital Archive Integrated Architecture.

The goal of the ***Unified Ontological Model*** upper layer, is getting the semantic integration of information whose characteristics have been discussed in [6]. It is composed of two levels: the **Specific DA Ontologies** (First Semantic Description Level) and the **Global Ontological Kernel** (High-Integrated Semantic Description Level).

At the first semantic description level, the ontologies stay in the pre-existent form. Only, mappings are established among them between concepts from one to another increasing the semantic scope.

The Global Ontological Kernel implementation will be carried out based on the mappings between ontologies. It merges these ontologies operating as a kernel in which current specialized ontologies could be semantically connected, as figure 2 represents. Thus, this ontological kernel is in charge of providing a common understanding for fundamental DA concepts.

This paper deals with the ***Data Extractor Model*** bottom layer, close to the back-end data sources, describing its lowest component, called ***Data Extraction i***. The ***Wrapper*** will be discussed in detail in future works.

2.1 Data Extractor Model

The data extraction is the task of identifying and extracting specific data fragments from a collection of documents presented as a query result [29, 32]. XML is the data exchange standard language, making possible interoperability and sharing data [28].

As figure 2 represents, the ***Data Extractor Model*** is the layer in charge of the information extraction from heterogeneous data sources. It is composed of two components: the ***Data Extraction i*** (***DE***) and the ***Wrapper***.

The *DE* overcomes the structural and format heterogeneity due to a particular data source content. In this way, one *DE* is integrated with one specific *Wrapper* tailored to one specific DA ontology. To reach this kind of integration, the *DE* translates (adapting and hiding syntactic heterogeneity) the information between one *Data Source* and the corresponding *Wrapper*.

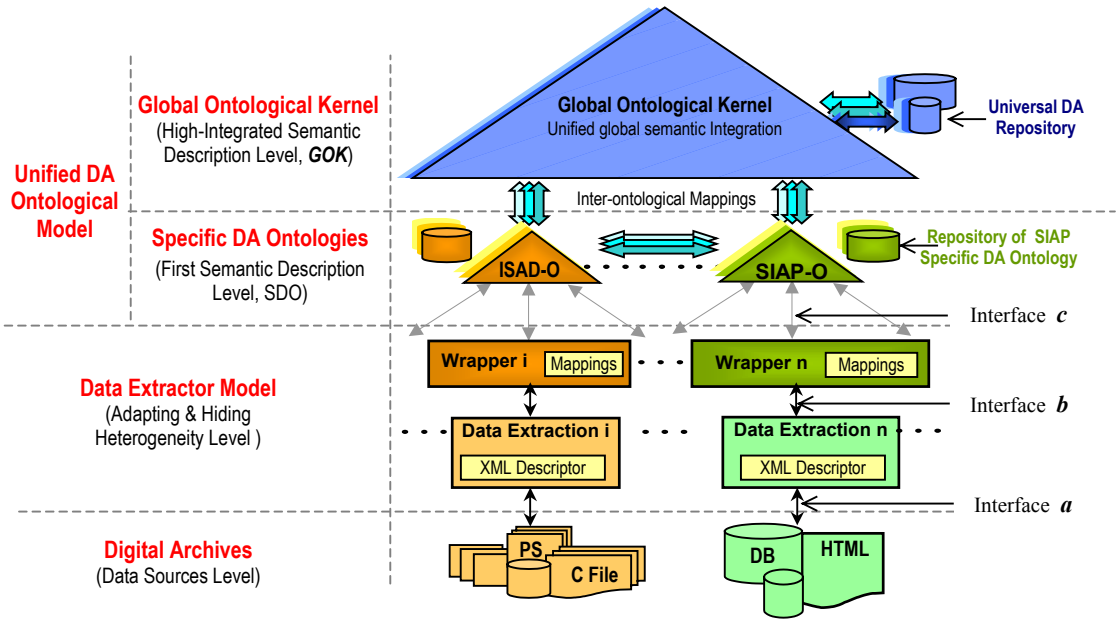

Figure 2. The Proposed Virtual Integration Model

The *Wrapper* is univocally associated to one specific ontology at the *Mediator* and overcomes the lexical heterogeneity of the XML terms coming from the *DE*. It makes mappings (logic links) between the ontological concepts and the syntactic terms coming from the *DE* [6]. Through these mappings it translates the queries and their results, transparently to the users.

Summarizing, the ***Data Extractor Model*** (*DE+Wrappers*) is in charge of linking the semantics among *Data Sources* and the *Mediator*. Because of this, the integration goal becomes possible: **invoking distributed queries against several data sources as if they were only one**.

3 Data Extractor Component (DE)

The DE component, above the DA sources, provides the Wrappers an uniform layer in order to invoke local data source queries. Through a XML Schema, Wrappers and data sources agree the common terms shared to exchange information [28]. In this way, the data heterogeneity is overcome at the format and structure of the content levels.

Each particular *DE* imports from the associated data source its structures and specifies its own integration capabilities. These capabilities are defined in a *XML Descriptor* document holding the following metadata:

- Structure and format of the source contents.
- Query capabilities: data range restrictions, enabled query terms, etc.
- XML Schema: as a local data source view.
- Specific Ontology, which the data source wants to be integrated to.

So, each particular *DE* establishes its role as a participant for each concise integration. Thus, the data sources impose their own restrictions in each kind of integration.

Using this XML Schema view, the *DE* component receives queries from the *Wrapper* and pre-processes them to invoke the local source queries. Additionally, two or more sources can integrate their XML schemata from several *XML Descriptors* and present them to the *Wrapper* as if they were only one. The *Wrapper* would define the query over the new integrated schema, which contains the restrictions imposed by the original schemata [22].

The *DE* component makes also possible to implement web services associated to the data sources and publish their data on the web as XML documents, avoiding the user's straight interaction with the data sources. For this reason, the *DE* can be running stand-alone as a web site with remote accesses to web data sources [20].

3.1 Data Extraction Architecture

Basically, as it is shown in figures 2 and 3, each DE carries out two main functions, corresponding to its two interfaces, described as follows:

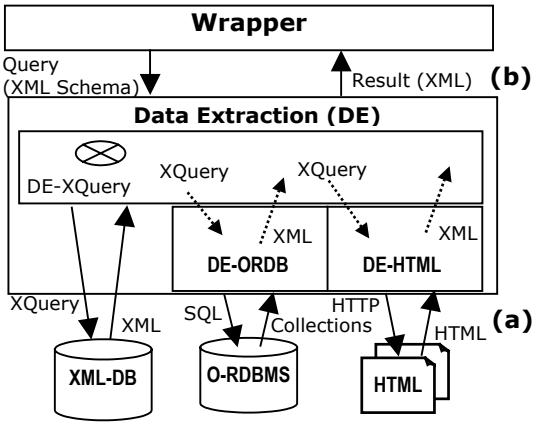

Figure 3. Data Extraction component architecture.

Interface a, *with the Data Sources***.** Its purpose is requesting local queries to the underlying data sources and extracting the results in XML format. This action is carried out in two steps:

1. Translating the XML Schema query, received from the *Wrapper*, to the source's query language (SQL/ XML, XQuery + XPath, etc). We use XQuery language for XML (defined by the W3C) as an intermediate step for querying the source.
2. Requesting the local source query, collecting the result and formatting it in XML. This XML is compliant with the schema received from the *Wrapper*. The way the *DE* gets the data result depends on the type of the underlying data source (from object-relational or XML databases to file servers with static HTML files, PDF files, etc).

Due to this data sources' diversity, this paper considers three scenarios:

- Relational and object-relational databases. In this case, it is necessary to translate the query into the language used by the O-RDBMS (SQL, SQL/XML, etc). The O-RDBMS returns the result as collections (objects) or sets (relations). As we know, this kind of results could be straightforward translated into XML.
- XML-DB. These kind of DBMS manage collections of XML documents (xIndice, http://XML.apache.org/xindice; eXist, http://exist-db.org; LegoDB, [2]). They can execute XQuery and XPath queries, so they do not need any additional pre-processing.
- HTML Web servers. There are lots of web sites with HTML static web pages. In order to integrate this kind of data sources it is mandatory to translate their HTML documents into XML.

In any other scenario, where the source does not match these three previous ones, the corresponding *DE* must conform to the particular source in order to extract the data as XML documents.

Interface b, *with the Wrapper.* The aim of this interface is to provide an uniform layer to query the data sources. This interface is specific for every *Wrapper*. It is in charge of hiding the structure and format sources content heterogeneity to the *Wrapper*. The *DE* presents a generic interface for returning XML results. It specifies the views of the sources through XML schemata. The *Wrapper* submits queries applying the user query restrictions to the related view XML schema.

3.2 Data Extraction Organization

In fact, the DE runs within each source at three stages:

1. Data Source Integration. For each source view, the integration is made by creating a *XML Descriptor* in its related *DE*. The *XML Descriptor* is a XML file containing information about: source type, integration capabilities of each view and data extraction rules [10]. All this information is stored in the following *XML Descriptor* items:

- *References (URIs).* They specify how to allow the navigation within the source: user, pass, grants, etc.
- *Local View (XML Schema).* It specifies the structure and content format (metadata) in a XML Schema, as a source view. Through this schema, the *Wrappers* can impose restrictions and create the query definitions. The schema holds information about the data structures (for example, the RDB tables) and defines each element by its data type, regular expression and possible restrictions (range, format, etc).
- *Style (XSLT, XSLT-FO).* This item includes the transformations applied to each element in the schema for the returning result in a visual form: HTML, PDF...
- *Mappings.* This item contains the mappings among the data source elements and the XML schema view elements. For each element, the mappings collect information about its name, location, etc.
- *Extraction Rules.* This item keeps the rules specifying how the query is invoked at the data source. These rules are adapted to any target particular source and can be defined as XSLT, SQL or XQuery, etc.

2. Query Translation. The wrapper sends the query to the *DE* as a XML schema. As an intermediate step, the *DE* translates this query into XQuery language. Finally, the *DE* invokes the query to the local data source. Currently, our developed tool performs this translation through a transformation provided by a style sheet in XSLT (figure 4). Once translated, it is possible to query XML databases through XQuery language.

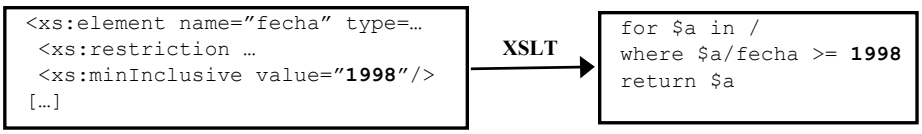

Figure 4. Simple example of the XML Schema into XQuery translation.

3. Invoking Local Queries. Finally, the query is executed at the local data source. Each kind of source imposes its own query execution way. So, the *DE* has to be adapted to each specific query processing. In Section 4, we describe the *DE* method applied to carry out local queries in HTML Web Servers, O-RDB and others. However, in the XML-DB sources, the *DE* invokes the query in the XQuery language and gets the result in XML. Anyway, the result, already in XML, will be returned to the related *Wrapper*.

4 DE adapted to a particular Data Source

Due to data sources' heterogeneity, each source requires different characteristics of its related *DE*. From the three possible scenarios here considered, this section deals with only two of them, because the XML-DB case does not need any additional query pre-processing inside the *DE*. Therefore, this section mainly describes the two other scenarios: HTML and O-RDB sources.

4.1 Extraction from HTML into XML

There is an important research activity about automatic and semi-automatic applications designed to perform the HTML extraction, as it is well discussed in [23]. According to [21], extracting structured data from HTML requires to solve five problems:

- *Navigation*: finding the HTML web pages through the hyperlinks, javascript, forms, etc.
- *Extraction*: identifying and extracting the relevant data in these HTML pages.
- *Structural*: providing the extracted data an output structure with a XML schema (content structure and data type).
- *Element Alignment*: ensuring the homogeneity of the data related to the output schema.
- *Data Integration*: merging the data obtained from different HTML pages in one unique XML document.

There are several kinds of applications dedicated to the HTML information extraction. Ones are based on trainings through examples as [19], or use heuristics as [31]. Others are based on patterns like Omini, Xwrap [3, 18], Lixto [17], Xtros [29]; or on extracting the data from HTML tables or lists as [12, 16].

According to its performance, you can distinguish among applications based on searching text information between marks (*LR, string wrappers*) or based on analysing the tree structure proper of the XHTML documents *(tree wrappers).* Taking into account all these valuable approaches, we propose here a data extraction procedure from the HTML, as represented in figure 5. First, we parse the HTML through its tree structure. After, we get the information of the content in each node of the tree as text. All the rules necessary for extracting the relevant data from the HTML will be stored in one *XML Descriptor.* To create a *XML Descriptor,* our methodology applies five steps in order to solve the five problems just said:

1. Tree parsing. First of all, we get the references of the web pages to parse (by using a crawler) and store them (URIs) in the corresponding item in the *XML Descriptor*. This solves the ***navigation*** problem. Secondly, we transform the HTML files into well-formed XHTML (by means of *W3C Tidy Utility,* http://www.w3.org/People/Raggett/tidy) and convert it into a tag tree representation, based on the nested structure of start and end tags.

Most web pages are designed for human browsing. Inside their HTML, there are specific branches dedicated to the style design for browsers (such as advertisements, menus, etc) and others that hold the content information. We move apart this content information sub-tree through algorithms like those described in XWrap [18]: *Highest Fanout* algorithm, compares the number of children for each sub-tree. *Largest Tag Count,* compares the number of tags in each sub-tree. *Greatest Size Increase,* compares the increase of the content between sub-trees. The sub-tree with larger fanout, tag count or/and greatest size, is the more likely to be the parent of the relevant data. (See the two trees on the right side in fig. 5)

Once identified the relevant sub-tree, we have to separate the children data objects inside it as smaller sub-trees. These child sub-trees are usually structured with a similar pattern and separated by the same tag. For each object sub-tree, the info nodes are included among HTML mark-ups (such as font tags) that do not carry any meaningful

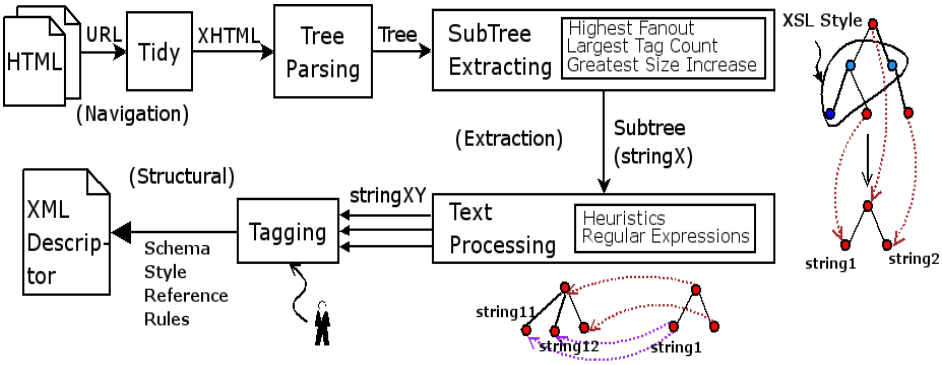

Figure 5. Proposed Workflow for the *XML Descriptor* creation

information. These HTML mark-ups are collected in XSL format in the *Descriptor* for each node of information. This process is made recursive until the tree contains only nodes with relevant information. This solves the ***extraction*** *problem*. Once cleaned, all nodes are XML tags including relevant content. The information tree structure gets mirrored into the *XML schema* that represents the output structure of the HTML content. This partially solves the ***structural*** problem.

Every transformation applied to the HTML tree keeps stored as XSLT transformations in the *Extraction Rules* of the *XML Descriptor*. Also, we keep the HTML tree path for each element of the schema in the *mappings* item. By using this path, you can establish correspondences between the information in the HTML tags and the XML schema elements. These rules allow a direct information extraction from HTML pages by applying these transformations.

2. Text processing. Till now, we have obtained the output tree from the data content structure. However, the format of each element's content is unknown. So, the info nodes in this tree are processed as strings, and their XML tags are renamed into *stringX*. These nodes tagged *stringX* carry information including patterns or data types as *substrings*. By means of regular expressions or heuristics, you can find the information elements in those *substrings*. For example, we can find patterns for dates, money, ISBN, etc. Taking the *stringX* nodes, we split them into new *stringXY* nodes holding all new elements found. Meanwhile, the nodes without any *substring* remain unchanged. In this way, this tree structure expands its leaves. (See the bottom two trees in fig. 5). The found patterns and data types keep held into the *XML Schema* as the *type* attribute for each new element described. So, all the transformations here described are also stored inside the *XML Descriptor*.

3. Tagging. At this step, we have to name each element *stringXY* in the schema, so that the tag name matches the content it represents. For example, a *stringXY* node containing a given date needs to be tagged as *date*. In order to automatically name one element (as right as possible), the elements have to be analysed in format and content. Obviously, in case of HTML tables with titles row (<TH>), these titles constitute the names assigned to the table elements. But in case of plain text it is more difficult, for instance: finding out that a word is the name and the following ones are the middle name and the surname. Our tool applies some algorithms to guess, whenever possible, the name of the elements (*Output Tagging* in [18]). These algorithms try to infer the tags from the element contents by comparing them against a repository, matching regular expressions (for example, dates, ISBN) and the appearance order (for example, money usually gets followed by €, $, etc). In case of it is not possible to find out proper names for the tags, the user has to do it manually, through some other semi-automatic application.

Once found and tagged all information nodes in the tree extracted structure, the *XML Schema* is complete. It contains the content structure, and the data format. The *XML Schema* contains the output structure from the web pages data and could be used as a source's content local *view*. Now, the ***structural*** problem is solved. The source's administrator can re-define these views applying other constraints on these *XML schemata*.

4. Alignment. The objects found in each HTML page can contain less elements than those identified in the output XML schema. In order to assure the right assignment of the information nodes matching with the schema, we compare the format and data type among nodes and elements before creating the resulting XML document. This solves the ***alignment*** problem.

5. Integration. The data integration of many web pages (compliant with the same output schema) takes place at the local query processing time. The elements extracted from each web page are aligned with its schema. The result is an unique returned XML document including the data obtained from all the source pages. This solves the HTML ***integration*** problem.

All necessary HTML extraction rules keep stored inside the *XML Descriptor*. With these rules, you can create the XML document containing all the source's view related information. The *XML Descriptor* holds information of each element about its XSL style, tree position (*path*), content and format.

As the HTML can change, it does not seem useful to generate the XML document with all the information and store it in a static context. However, by using the *extraction rules* (stored in the *XML Descriptor*) you can do the data extraction *on-the-fly* when the query is submitted. Thus, data always remain at the source and they are not replicated in other XML documents. This technique is usually known as *virtual integration* (non *materialized*). So, data consistency between *DE* and the sources is always guaranteed.

As a consequence of the described methodology, and considering that the *extraction rules* are stored in the *XML Descriptor* (as XSLT transformations) and that the query is defined in XQuery language at the *DE*; we think that an interesting future work could be the direct application of these transformations to the query instead of HTML pages. In this way, the query could be executed directly by the *DE*'s XQuery processor over the XHTML web pages.

4.2 Extraction From Object Relational Databases to XML

It is well known that current SQL-99 (XSQL, SQL/XML, etc) extracts data from any O-RDB. In the last decade, many O-RDBMS have appeared merging both XML and SQL query languages (Oracle8i and successors, DB2-XML, etc).

In our proposed architecture, the *DE* receives (from the *wrapper*) the query defined by XML schema restrictions and translates it into XQuery. So, we need to translate this query into SQL.

The task of translating XQuery into SQL depends on the affinity between XML and SQL in the source's query processing. The most difficult situation is when the source query language follows SQL-92 (and ancestors). For this hard situation, several works have carried this topic out: Xperanto [27], SilkRoute [24], Xtables [1,9,14]. In order to materialize the translation we need considering two main issues: the election of the XML schema representing the relational structure of the data [13, 27] and the query translation itself.

Concerning to the first issue, the XML Schema represents the view of the relational schema from the XML model point of view. Following a GAV (*Global As*

View) approach [4], the *DE* holds the *XML Schema* into the *XML Descriptor,* created at the integration moment. It represents the source view, now over the O-RDB tables.

For the second issue, the *DE* makes a direct translation from the XML schema query into SQL, avoiding the XQuery translation. The *mappings* item in the *XML Descriptor* stores the correspondences among the XML schema elements and the RDB tables' columns. Also, they store the relationships among those table columns (to make easy the translation of *joins*, *keys*, etc). Additionally, the use of annotations in the schema is also considered (as in SQL Server) or taking profit of the SQL/XML capabilities.

Figure 6 shows an example of the mappings creation and the query translation into SQL. In it, the arrow numbers indicate the extraction and translation steps.

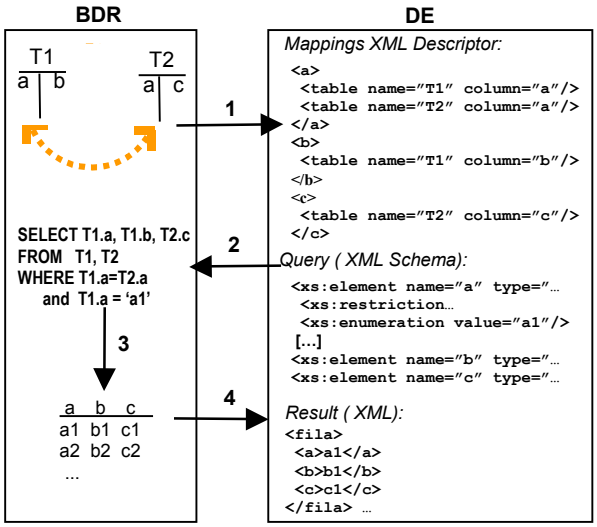

Figure 6. Example of extraction from RDB.

In (1) the mappings are created and stored into the *XML Descriptor*. After, when a query is received from the wrapper, the XML schema specifies the query by restrictions that have to be translated into SQL. In (2), the *DE* translates the query into SQL and makes use of the mappings to match each schema element with the equivalent RDB column. In (3), the SQL query is executed, returning the results as table sets. Finally, in (4), using the mappings again, the XML document is created as the final local results.

4.3 Data Extraction From Other Sources

Current web information adopts several formats. Each *DA* and each possible format needs a particular *DE*. This section considers that, in the future, the *DE* will need a wider scope than the one just described. So, other kinds of sources, as PDF and Microsoft Office files, can be also included for querying.

PDF format is widely extended for the document diffusion in the web. PDF preserves the font-style, graphics and layout of source documents during electronic dis-

tribution and prevents subsequent editing by recipients. PDF is one of the '*de facto*' standards for sharing information between people. Unlike other document formats, PDF content is very difficult to parse and, hence, is extremely difficult to automate content extraction from a PDF document and convert it into XML. To extract PDF files, there are some proposals [30] and tools like Apache-FOP (http://XML.apache.org/fop), CambridgeDocs PDF XML (http://www.cambridgedocs.com) and ADOBE XMP.

Also, Microsoft Office format documents are having widespread acceptance nowadays. There are some projects trying to parse this kind of documents and extract its content into any other format: Jakarta POI (http://jakarta.apache.org/poi/, a Java API to Access MS Format Files) and CambridgeDocs, among others.

We are thinking, as future works, about the development of the *DE* additional specific modules adapting each kind of source documents.

5 DE General Overview

This section gives a general overview about the operating DE. As figure 7 represents, firstly the Wrapper requests the view schema to the DE. This schema is included in the XML Descriptor. The Wrapper defines the restrictions imposed by the user and submits the XML Schema query to the DE. The DE receives the query and translates it into XQuery in order to invoke its underlying source execution. As we have told, the DE is in charge of invoking the local query and collecting the returning results. The Wrapper receives them as a XML document. If the source were a XML-DB, then the DE simply executes the query through XQuery language.

In case that the source were an O-RDB, the query is translated into SQL using the *mappings* of the *XML Descriptor*. The *DE* invokes the local SQL query and collects the results as a table. Using the *mappings* again, the *DE* creates the final result as a XML document, which is returned to the *Wrapper*.

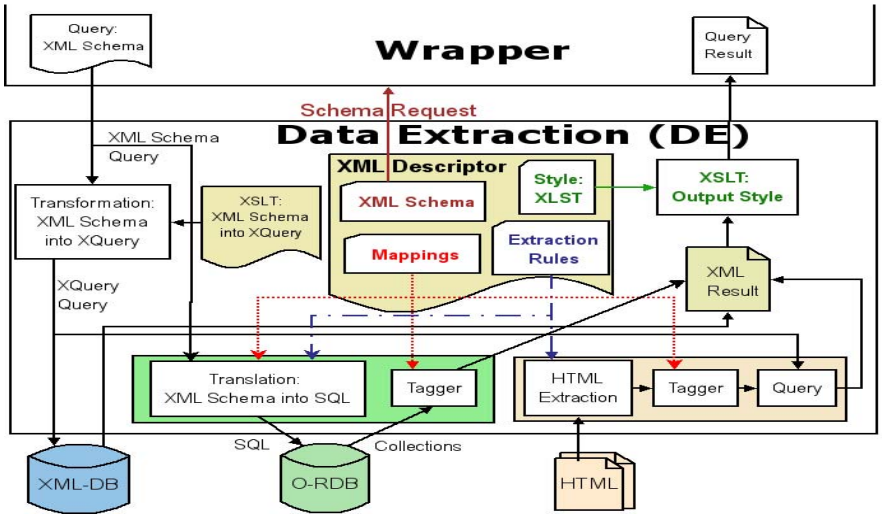

Figure 7. *DE* operating architecture.

If the source were composed of HTML web pages, the *DE* uses the item *references* to navigate through the collection of documents. Later, applying the *extraction rules*, the *DE* extracts the HTML information and creates the XML document according to the output schema. Then, the *DE* executes the query (in XQuery) and returns the results formatted with the style defined in the *XML Descriptor*.

In brief, all the semi-automatic procedure requires minimal user interaction in the data source integration. The user only has to specify the unknown tags in the output schema for HTML data sources. Additionally, the user can modify or restrict the views over the source by updating the XML Schema present in the Descriptor.

6 Conclusions and Future Work

This paper has introduced the **Data Extractor Model** inside a virtual and dynamic web integrated architecture for multiple heterogeneous Digital Archives data sources.

Belonging to this model, we have described in depth the ***Data Extraction (DE)***, a main component in charge of hiding any particular syntactic heterogeneity (format and content structure) coming from any digital archive. This *DE* is close to the back-end data sources and provides, at the bottom abstraction level, a first syntactic uniform layer to the upper ones.

As key issues involved in extracting and translating heterogeneous data, the paper has discussed how the *DE* interacts depending on its related source nature. This *DE* includes the different modules for extracting HTML web pages, XML-DB and Object-Relational Database web sites, as the most widely used in the current web.

The research and development are now focused on concluding the construction of the *DE* tool. Some future research lines are also focused on the extraction of information from other source formats, such as PDF or MS-Office, and their development as *DE* modules.

Acknowledgements

This work is partially granted by Spanish Ministry of Science and Technology (MCYT-TIC2002-04050-C02-02, DAWIS-UPM project), by Community of Madrid (07T/0056/2003/3, EDAD-UPM project) and by Spanish NoE Databases, RedBD (MCYT-TIC2000-3250-E).

References

1. Almarimi A, *Querying Heterogeneous Distributed XML Data*, in Sixth International Baltic Conference on Data Bases and Information Systems, Riga, pp. 177-191. 2004
2. Bohannon P, Freire J, Haritsa J, Ramanath M, Roy P, Simeon J, *Bridging the XML-Relational Divide with LegoDB: A Demonstration*, Proc. of the 19th International Conference on Data Engineering (ICDE'03), 2003

3. Buttler D, Liu L Pu C, *A fully Automated Object Extraction System for the World Wide Web*, in The IEEE 21st International Conference on Distributed Computing Systems, Phoenix (Mesa), Arizona, 2001.
4. Chawathe S, García-Molina H, Hammer J, Ireland K, Papakonstantinou Y, Ullman J and Widom J, *The TSIMMIS project: Integration of heterogeneous information sources*, Proc. tenth IPSJ Conference, Tokio, October 1994.
5. Costilla C., Calleja A. and Cremades J, *SIAP: Sistema de Información para Ayuntamientos y Parlamentos*, Revista Círculo de Usuarios de Oracle, CUORE, Sección 'Vivat Academia', Oct. 2003.
6. Costilla C, Palacios JP, Rodríguez MJ, Cremades J, Calleja A, Fernández R and Vila J, *Semantic Web Digital Archive Integration*, in Proc. Int. Workshop on Web Semantics (WebS 2004), 14th Int. Conf. on Database and Expert Systems Applications, DEXA2004, ISBN: 0-7695-2195-9, pp. 179-185, Zaragoza, Spain, Sept. 2004.
7. Costilla C, Palacios JP, Rodríguez MJ, Fernández R, Cremades J and Calleja A, *Web Digital Archives Integrated Architecture*, in the 5th International Conference on Internet Computing (IC 2004), The 2004 International MultiConference in Computer Science & Computer Engineering, Las Vegas, 2004.
8. Costilla C, Rodríguez MJ, Palacios JP, Cremades J, Calleja A and Fernández R, *A Contribution to Web Digital Archive Integration from the Parliamentary Management System 'SIAP'*, Proc. of Sixth Int. Baltic Conf. on Data Bases and Information Systems (DB&IS'2004), Barzdins J (ed.), ISBN:9984-770-11-7, pp. 481-496, Riga, Latvia, June, 2004, http://www.riti.lv/dbis2004.
9. DeHaan D, Toman D, Consens M, Ozsu M.T, *A Comprehensive XQuery to SQL Translation using Dynamic Interval Encoding*, in ACM Sigmod-Record, 31(2), June 2003.
10. Ek M, Hakkarainen H, Kilpeläinen P, Kuikka E, Penttinen T, *Describing XML Wrappers for Information Integration*, in XML Finland 2001 Conference "Surviving the XML (R)evolution", November 14-15, Tampere, pp. 38-51, 2001.
11. Eibe S, Costilla C, Menasalvas E y Acuña C, *DAWIS: Una Arquitectura de Integración Web para el Acceso Integrado a Archivos Digitales*, VIII Jornadas de Ing. del Software y Bases de Datos, pp. 583-591, Alicante, Spain, 2003.
12. Embley D, Tao C, Liddle S, *Automatically Extracting Ontologically Specified Data from HTML Tables of Unknown Structure*, in The Entity-Relationship Approach (ER 2002), pp. 322-337, 2002.
13. Elmasri R, Wu Y, Hojabri B, Li C and Fu J, *Conceptual Modelling for Customized XML Schemas*, in The Entity-Relationship Approach (ER 2002), pp. 429-443, 2002.
14. Fan C, Funderburk J, Lam H, Kiernan J, Shekita E, Shanmugasundaram J, *XTABLES: Bridging Relational Technology and XML*, IBM Systems Journal Volume 41, Issue 4 (October 2002), pp. 616 – 641, 2002
15. *The Mellon Fedora Project: Digital Library Architecture Meets XML and Web Services*, IV European Conf. on Research and Advanced Technology for Digital Libraries, Rome, Sept. 2002.
16. Gao X, Sterling L, *AutoWrapper: automatic Wrapper generation for multiple online services*, in Asia Pacific Web Conference, Hong Kong (1999).
17. Gottlob G, Koch C, Baumgartner R, Herzog M and Flesca S, *The Lixto Data Extraction Project: Back and Forth between Theory and Practice*, in PODS'04, Paris, June, 2004.
18. Han W, Buttler D, Pu C, *Wrapping Web Data into XML*, in ACM SIGMOD Record, 30(3), 33-38, September 2001
19. Ikeda D, Yamada Y, Hirokawa S, *Expressive Power of Tree and String Based Wrappers*, in WS on Information Integration on the Web (IIWeb-03)
20. Iturrioz J, Díaz O, Anzuola S, *Facing document-provider heterogeneity in Knowledge Portals*, 16th Int. CAISE, pp. 384-397, Riga, Latvia, June 2004.

21. Myllymaki J., *Effective Web Data Extraction with Standard XML Technologies*, Proceedings of the tenth International Conference on World Wide Web International, 2001, Hong Kong, pp. 689 - 696 ISBN:1-58113-348-0
22. Lee M, Yang L, Hsu W, Yong X, *Xclust: Clustering XML Schemas for Effective Integration*, in Conference on Information and Knowledge Management (CIKM'02), p. 292 November, McLean, Virginia, USA, 2002.
23. Laender A, Ribeiro-Neto B, da Silva A and Teixeira J, *A Brief Survey of Web Data Extraction Tools*, in ACM Sigmod Record, 31(2) pp.84-93, June 2002.
24. Fernandez M, Wang-Chiew Tan, and Suciu D,. *Silkroute: Trading between relations and XML*, Proc. Ninth Int. WWW Conf., 2000, Amsterdam
25. Open Archives Initiative, *Implementation Guidelines for the Open Archives Initiative Protocol for Metadata Harvesting Protocol*, Version 2.0 of 2002-06-14 Document Vers. 2002/06/13T19:43:00Z, 2002.
26. Sáenz J, Costilla C, Marcos E y Cavero J, *Una Representación en UML del Metamodelo Estándar ISAD(G) e ISAAR(CPF) para la Descripción de Archivos Digitales*, VIII Jornadas de Ing. del Software y Bases de Datos, JISBD'03, pp. 519-528, Alicante, 2003.
27. Shanmugasundaram J, Kierman J, Shekita E, Fan C, Funderburk J, *Querying XML Views of Relational Data*, in Proceedings of the 27th VLDB Conference, pp. 261-270, Rome, Italy, 2001
28. Vianu V, *A web Odyssey: from Codd to XML*, in ACM SIGMOD Record 32(2), pp. 1-15, June 2003.
29. Yang J, Choi J, *Knowledge-Based Wrapper Induction for Intelligent Web Information Extraction*, in [32], pp. 153-171, Springer-Verlag New York Inc., 2003
30. Yonggao Yang, Kwang Paick, Yanxiong Peng, Yukong Zang, *PDF2XML: Converting PDF to XML*, Proceedings of the 2004 International Conference on Information and Knowledge Engineering (IKE04), Las Vegas, 2004
31. Yip Chung Ch, Gertz M, Sundaresan N, *Reverse Engineering for Web Data: From Visual to Semantic Structures*, in 18th International Conference in Data Engineering, San Jose, California, 2002
32. Zhong N, Liu J, Yao Y (eds.), *Web Intelligence*, Springer Verlag, 2003

Processing Queries over RDF Views of Wrapped Relational Databases

Johan Petrini and Tore Risch

Department of Information Technology, Uppsala University, 75195 Uppsala, Sweden
{Johan.Petrini, Tore.Risch}@it.uu.se

Abstract. The semantic web standard RDF enables web resources to be annotated with properties describing structure and contents. However, there is a vast amount of additional high quality information in the *hidden web* stored in databases accessible from the web but not as web pages. In particular, most organizations use relational database technology and such databases should also be accessible from semantic web tools. We are developing a system to transparently wrap relational databases as *virtual* RDF resource descriptions. The wrapper provides relational database access to the Edutella infrastructure for searching educational resources. Semantic web queries are expressed using a Datalog-based RDF query language that through the wrapper transparently retrieves information from relational database servers. Since the data volume in these databases is huge, the data cannot be downloaded but instead virtual RDF resource descriptions are returned as query results. Query optimization techniques permit transparent and efficient mapping from RDF queries to relational database queries. Semantic web queries are more dynamic than relational database queries and they may freely mix access to data and schema. This makes it necessary to optimize not only data access time but also the time to perform the query optimization itself.

1 Introduction

Modern information systems often need to access many different kinds of data, including Internet-based web resources, relational databases, or files containing experimental results. It is getting increasingly difficult to get the correct information when retrieving information from web resources using traditional unstructured free-text based search methods as provided by, e.g., GOOGLE.

Normally the useful information is hidden inside huge amounts of irrelevant information. This data retrieval problem gets even worse if one wants to combine web resources with other kinds of data stored outside the web, for example in enterprise databases, often referred to as the *hidden web*. Either one has to manually browse-cut-and-paste between web search tools and database tools, or one has to develop hard-wired programs accessing data from relational databases and web sources for combining and filtering the retrieved resources.

The semantic web initiative [3] aims at providing Internet-wide standards for semantically enriching and describing web data. Using the standards RDF [7][14] and

RDF-Schema [4], abbreviated as RDFS, any web resource can be annotated with *properties* describing its structure and contents. This facilitates guided search of web resources in terms of these properties. The properties are represented as sets of RDF *statements*, which are triples containing a web resource (the *subject*), a property (the *predicate*), and a value (the *object*). RDFS [4] adds semantics to basic RDF with schema definition capabilities providing, e.g. classes, inheritance, and restrictions on the kinds of properties a given class of web resources can have. RDF is used, e.g. by the Dublin Core standard [8] for meta-data markup of library data and the Edutella infrastructure [18] uses it for searching educational web resources.

Queries to semantic web data are specified using some of the query languages proposed for this, e.g. RDQL [23], RQL [13], and QEL [21].

We are developing a system SWARD (Semantic Web Abridged Relational Databases) for scalable RDF based wrapping of existing relational databases in the *hidden web*. Instead of downloading the relational database tables into RDF repositories we map an existing relational database schema into a corresponding *virtual* RDF statement set of the wrapped relational database. When semantic web queries reference this virtual statement set the system automatically translates fragments of the queries into one or several SQL queries to the relational database. The result of a query is not explicitly stored as RDF statements in a repository, but statements are instead dynamically generated as data is retrieved from the relational database. Query filters in RDF queries are moved into SQL selections when possible. Filters not expressible in SQL are applied on the results from the SQL queries. A particular problem is that queries to RDF statement sets do not need to distinguish between what is schema and what is data as in relational databases. In RDF both schema and data are mixed in the statement sets and, unlike SQL, queries to RDF sources do not need to be expressed in terms of a database schema. This prohibits pre-compilation of queries as in SQL.

The approach is evaluated initially in the context of the Edutella framework [18], which is a peer-to-peer infrastructure for semantic web searches of educational resources. Edutella uses the query language QEL [21], a Datalog [27] based query language for RDF. Each educational source made available to Edutella is called an Edutella *provider*. A provider receives dynamic QEL queries from Edutella to a specific source. It evaluates the query and returns the result as a set of RDF statements. Our provider permits QEL queries to any wrapped relational database. As test case we provide RDF query access to a relational database storing information about Swedish museums, *Museifönstret*[1][17]. Since, unlike SQL, Edutella queries are always dynamic and cannot be precompiled, we optimize not only the query execution time as relational databases but also the query compilation time as is the focus of this paper.

Our approach enables efficient semantic web peer-to-peer queries to the combination of resources on the web and in the *hidden web*. It allows Edutella peers to access existing relational databases as well as other sources, even though the relational databases have totally different data representations than RDF-statements. The system manages mappings between meta-data descriptions based on Dublin Core used in the Edutella ontology and the wrapped relational databases. The system furthermore allows user functions implementing algorithms for data filtration, indexing, and fusion, and it enables transparent use of these user-defined functions in queries and views.

[1] English:'The Window to Museums'.

2 Related Work

Usually, RDF statements are stored as web documents or in internal relational databases designed for RDF [2][26]. The schema of the relational database is internal to the repository system. One problem with storing all data as triples is that the repository does not have any knowledge about the most efficient representation for each application, which makes relational query optimization less effective. To alleviate this, Jena2 [26] uses property tables for non-triple representation of RDF statements.

However, if one wants to access existing large relational databases through semantic web queries using an RDF repository one needs to download the relational database tables into the RDF repository before querying them. This can be very costly if the relational database is large. By contrast regular relational databases are designed and tuned for maximal efficiency of the applications using the database. To limit data transmission they are designed for keeping all data in the database and only export a minimal amount of data for answering queries. The database schema provides application-oriented meta-data descriptions, efficient data representation, and efficient query processing.

Rather than storing RDF data in dedicated RDF-repositories our work wraps existing relational databases to be used in the semantic web queries without downloading database tables to a repository. Instead the statements necessary for answering a particular query are represented as virtual statements streamed through the wrapper. The closest works are D2R MAP [5], and RDF Gateway [20], which provides conversion methods from relational databases to RDF. The typed RDFS-based view specification language RVL [16] is proposed for semantic web integration [6]. However, none of the works deal with how to actually optimize semantic web queries over wrapped relational databases, the main topic of this work.

Several mediator projects [10][11][12][15][19][22][25] wrap external data sources from virtual databases. However, none of these projects deal with wrapping relational databases under semantic web infrastructures.

There are a few proposals for query languages for RDF e.g. RDQL [23], RQL [13], and QEL [21]. These query languages are based on declarative queries to the space of triples constituting an RDF database. In this project we primarily use QEL but the technique can be applied on the other query languages as well.

SWARD generalizes the Edutella peer-to-peer infrastructure [18] for searching learning materials on the web to permit providers to execute QEL queries over the *hidden web*.

3 Example

The relational database *Museifönstret* [17], abbreviated as *WM,* stores data about artifacts in Swedish museums. For example, a relation `Resource` in *WM* contains information about artifacts such as for example their name, description, URI and ID according to the schema:

```
Resource(RID,MID,Name,URI,ShortDesc,Desc)
```

RID is a numeric resource identifier and MID is a numeric museum identifier. Our Edutella provider SWARD wraps *WM* to appear as a set of RDF statements.

An example QEL query, *q1*, submitted to SWARD from Edutella is to find all museum artifacts with a name that contains the string 'Matter'. It is expressed in QEL as[2]:

```
@prefix qel:http://www.edutella.org/qel#
@prefix dc:http://purl.org/dc/elements/1.1/
?(x,t):-qel:s(x,'dc:title',t),
        qel:like(t,'Matter')
```

Here we use a Datalog-like syntax for the QEL query [21]. In practice it is sent from Edutella to the provider using a less readable equivalent XML syntax.

`qel:s(x,p,t)` is true if there is an RDF statement matching the triple `<x,p,t>` where `x`, `p`, and `t` are variables bound to resources. `t` may also be a literal. `qel:like(t,'Matter')` is true if either `t` is a literal and the string value of `t` contains the string `'Matter'`, or `t` is a resource and the URI of `t` contains the string `'Matter'`.

q1 is intentionally chosen to be very simple to enable, for the reader, a perceivable step-by-step translation of the query to SQL in later sections. However, in an experiment measuring processing time of a QEL query in SWARD, described in section 6, a more complicated query containing a join is used.

4 System Architecture

Fig. 1 describes the architecture of SWARD. A QEL query arrives at the SWARD Edutella provider. There is a *query statement generator* building on Jena2 that parses incoming RDF data serialized as XML and extracts existing RDF statements expressing QEL queries. The dotted arrow in Fig. 1 from the query statement generator to the *calculus generator* indicates logical flow of execution. Actually, the parsed RDF statements are first stored in the local *statement repository*. The calculus generator then translates the materialized RDF statements corresponding to a specific QEL query into a *domain calculus expression*. It includes a fix-point rewrite algorithm to minimize the domain calculus expression, before it is translated by the *cost-based query optimizer* into an algebraic expression. This algebraic expression is then interpreted by the algebra interpreter over the combination of *materialized* RDF(S) statements, explicitly imported and stored by the *statement importer* in the statement repository, and *virtual* RDF statements generated by the *relational statement wrapper*.

[2] @prefix notation used to abbreviate namespaces.

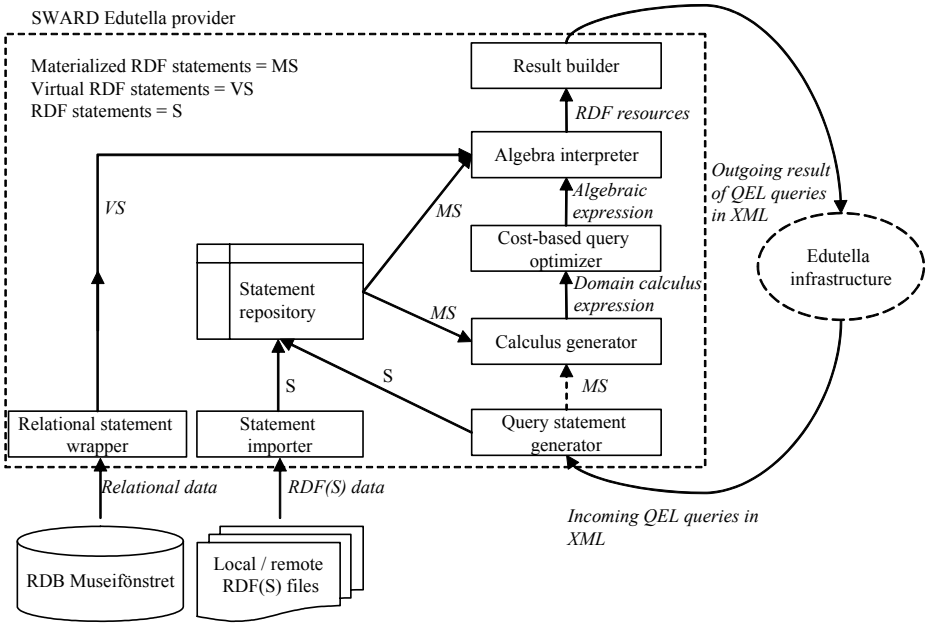

Fig. 1. Architecture of SWARD

Thus materialized statements form a local database in SWARD while virtual statements are views of data in external sources. Instances of virtual statements are dynamically created and streamed through the system. A garbage collector automatically removes no longer needed virtual statement instances. Finally, the result of executing the algebraic expression is sent back over Edutella as variable-resource or variable-literal bindings serialized as XML by the *result builder*.

Examples of materialized statements are statements imported from files containing RDF(S) data such as the W3C definition of RDF(S) as meta-data statements. Hence, as illustrated in Fig. 1, data in SWARD statement repository can originate from local or remote RDF(S) files or the Edutella infrastructure.

Fig. 2 illustrates the modeling of wrapper data sources in SWARD. SWARD extends the basic RDFS model with RDFS classes representing different *statement sources*[3], and the possibility to define a hierarchy of such sources.

An RDFS class acting as a statement source is instantiated only once by the system upon initialization. Each statement source has an associated property, `stmts`, maintained by the system that generates the statement set of RDF statements in the source. Notice that statement sets belong to a source while the resources referenced by the statements are universal.

[3] A statement source is a data source with its content translated into RDF statements. Observe that a RDFS class representing a statement source is modeling a data source and not the semantics of data in that data source.

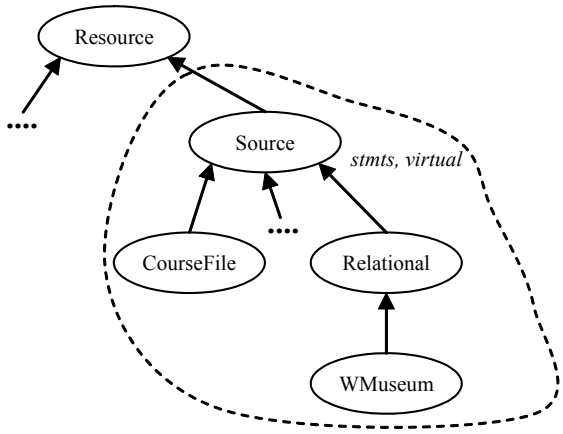

Fig. 2. Hierarchy of wrapper data sources in SWARD

There is a hierarchy of statement sources to handle that some sources are specializations of other, e.g. `WMuseum` is a specialization of general relational database statement sources. Statements in a statement source, *s*, are seen as a union of statements in statement sources subclassing *s*. The root class `Source` represents all RDF statements. Each other RDF statement source is a subclass of `Source`. There is a subclass to `Source` called `Relational` representing all relational database statement sources. As illustrated by statement source `CourseFile` we also allow other kinds of sources than relational databases. `CourseFile` represents RDF files containing courses read by computer science students at Uppsala University. This paper focuses on relational data sources which are subclasses to class `Relational`. For example, class `WMuseum` represents the specific relational statement source *WM*. This separation between `Relational` and its subclasses enables us to generate and compute tailor made statement sets for different databases. For example, often, for scalability reasons, SWARD should treat statements in a statement source representing a relational database as virtual statements. Therefore RDFS classes representing statement sources have a Boolean valued property `virtual` indicating if the statements belonging to a source are virtual or not. For the RDFS class to be virtual all its subclasses have to be virtual. The statement hierarchy can easily be extended with additional statement sources.

5 RDF Views over Relational Data

As illustrated in Fig. 1 a QEL query is translated from XML to an intermediate domain calculus representation. For each table $T(C_1,\ldots C_n)$ in relational database *R* where column named C_1 is key for simplicity, SWARD generates a set of views denoted $CSS(R, T, C_i)$ with definitions:

$$\begin{aligned} &CSS(R,T,C_i): \\ &\{s,p,o \mid s=uriKey(R,T,c_1) \wedge \\ &p=uriCol(R,T,C_i) \wedge T(c_1,\ldots,c_n) \wedge o=c_i\} . \end{aligned} \quad (1)$$

where *uriKey(R, T,* c_1*)* computes a unique URI for the key c_1 and *uriCol(R, T,* C_i*)* denotes a unique URI for the column named C_i. Notice that *CSS(R,T,*C_i*)* describes a column C_i, i.e. it is not materialized. The name of the view is generated by concatenation, e.g. `CSS('WM','Resource','Name') = WMResourceName`. Table I shows how URIs representing data from a relational database are auto-generated by the system, given a user defined namespace, *ns*[4].

Table 1. Schema for autogenerating URIs representing data from a table $T(C_1,\ldots,C_n)$ in a relational database *R* using namespace *ns*

Function	Generated URI
uriCol	*ns:databasename.relationname.columnname*
uriKey	*ns:databasename.relationname.columnname keyvalue*
compUriKey	*ns:databasename.relationname.* $columnname_1 \ldots columnname_n$. $keyvalue_1 \ldots keyvalue_n$

There are cases when URIs should be user specified rather than automatically generated. For this SWARD allows the user to explicitly specify *uriKey* for a CSS. For example, the table `Resource` in *WM* already includes a field, `URI`, containing unique URIs for each row (a secondary key) and this column is therefore chosen by the user to represent the *uriKey* in the definition of `WMResourceName` as illustrated in Example 1. Notice that the term `Resource` represents the wrapped relational table:

```
WMResourceName(s,p,o):

{s,p,o|
s=uri∧                          /*uriKey*/
p='ns:wm.resource.name'∧        /*uriCol*/
Resource(rid,mid,name,uri,shortdesc,desc)∧
o=name}
```

Example 1. Definition of WMResourceName

The algebra generator will combine CSSs appearing in a QEL query and generate SQL strings for accessing the wrapped database. Values from column named C_i are treated as literals if the column is not a foreign key. If C_i were a foreign key from another table in *R*, *T'*, the system would replace '`o=name`' in Example 1 with '`o=uriKey(R,T',name)`'. (Compound keys are treated as tuples and handled by the function *compUriKey*).

[4] Namespace:'http://www.museifonstret.se/'

The statement set, `stmts`, of a statement source that represents a relational database is defined as the union of all column statement sets for the source.

Fig. 3 illustrates how the statement set of `WMResourceName` is defined from column `Name` in table `Resource` in *WM*.

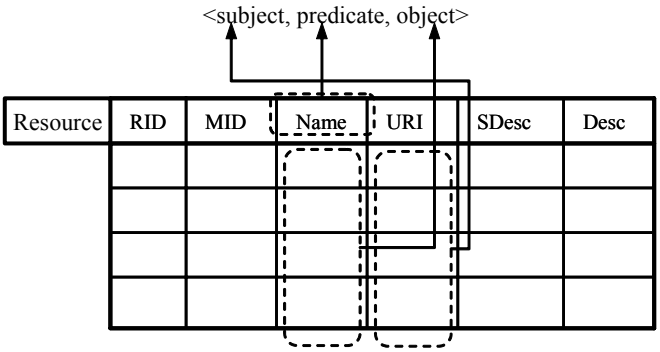

Fig. 3. Producing the statement set of `WMResourceName`

SWARD allows for different terminologies in received QEL queries and in URIs from the CSSs. In our example Edutella uses a terminology based on Dublin Core, which is different from the terminology of statement source `WMuseum`. The relations between two URIs from different terminologies having the same meaning are called *source mappings*. They are represented by a user defined table `SM(U1, U2)` in the wrapper taking two URIs `U1` and `U2` as arguments and evaluating to true if there is a source mapping between them.

For example, the RDF predicate produced by `WMResourceName` in Example 1 is mapped to the URI '`dc:title`' in Dublin Core.

6 Translation of QEL to Optimized SQL

QEL queries are specified against an RDF view containing both materialized statements from SWARD statement repository and virtual statements mapped from the wrapped relational database. In our example, *q1* is represented by the following domain calculus expression:

$$\{s,p,o \mid stmt(s,p,o) \wedge like(o,\text{'Matter'}) \wedge p=\text{'dc:title'}\}$$

`stmt` and `like` implements the built-in QEL predicates `qel:s` and `qel:like`, respectively. `stmt` is evaluated over the statement set *w* of statement source `Source`. `stmt(s,p,o)` evaluates to true if there is an RDF statement <s, p, o> in `stmts` of `Source`.

In the rest of this section, for simplicity, *w* is assumed to be equal only to RDF statements in `WMuseum`. Furthermore, *WM* is restricted to contain only one table,

Resource. Hence *w* can be seen as the disjunction of all CSSs in WMuseum and *q1* is expanded to the following expression:

```
{s,p,o|
(WMResourceRID(s,q,o)∧SM(q,p))∨
(WMResourceMID(s,q,o)∧SM(q,p))∨
(WMResourceName(s,q,o)∧SM(q,p))∨
(WMResourceURI(s,q,o)∧SM(q,p))∨
(WMResourceShortDesc(s,q,o)∧SM(q,p))∨
(WMResourceDesc(s,q,o)∧SM(q,p))∧
p='dc:title'∧like(o,'Matter')}
```

The resulting calculus expression is transformed into a simpler one by a fix point algorithm using rewrite rules [9]. Thus the above expression is first translated to *disjunctive normal form*:

```
{s,'dc:title',o|
(WMResourceRID(s,q,o)∧SM(q,'dc:title')∧
like(o,'Matter'))∨
(WMResourceMID(s,q,o)∧SM(q,'dc:title')∧
like(o,'Matter'))∨
(WMResourceName(s,q,o)∧SM(q,'dc:title')∧
like(o,'Matter'))∨
(WMResourceURI(s,q,o)∧SM(q,'dc:title')∧
like(o,'Matter'))∨
(WMResourceShortDesc(s,q,o)∧SM(q,'dc:title')∧
like(o,'Matter'))∨
(WMResourceDesc(s,q,o)∧SM(q,'dc:title')∧
like(o,'Matter'))}
```

Example 2. Expression of *q1* on disjunctive normal form

Each CSS is substituted for its definition and simplified. For the CSS WMResourceName, according to Example 1, this produces the following term in the disjunction above:

```
Resource(rid,mid,o,s,shortdesc,desc)∧
SM('ns:wm.resource.name','dc:title')∧
like(o,'Matter')
```

Example 3. Substitution of WMResourceName for its definition

With *compile time evaluation* the SM table is then evaluated by the *calculus generator* and replaced with TRUE since SM maps 'dc:title' to 'ns:wm.resource.name'. For the other terms in the disjunction (shown in Example 2) SM will be evaluated to FALSE and they will be removed. The only remaining calculus term is then translated into an algebraic expression, or query plan, by the *cost-based query optimizer* that contains calls to a foreign function executing SQL ac-

cording to Fig. 4. This shows that compile time evaluation substantially reduces the size of the calculus expression that is sent to the *cost-based query optimizer* and therefore reduces the query optimization time.

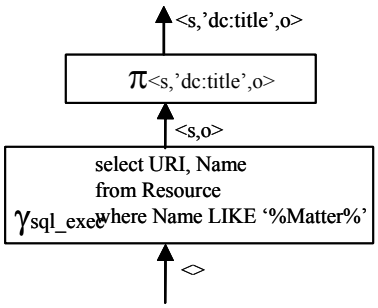

Fig. 4. An execution plan for the QEL example query

An algebra operator is one of {π, σ, ×, ∩, ∪, ⋈, γ}[9]. The π, σ, ×, ∩, ∪, and ⋈, operators have the same semantics as their relational counterparts. The γ (generate) operator performs function application. It can thereby introduce objects other than those produced by the leaf nodes into the query plan. In this way, the γ operator is similar to the generate operator in [24]. The function `sql_exec` sends an SQL query to a relational database.

An essential technique for improving the efficiency is to push down filter operators such as `like` to SQL when possible as in Fig. 4. The system has special rewrite rules for generating SQL strings from the calculus and it knows what functions can be executed in the sources, e.g. `like`, and generates SQL strings with calls to such functions. Furthermore, SWARD uses the heuristics to generate from a term as few SQL queries as possible but never a Cartesian product or a union.

In *q1* the variable `p` in the built-in predicate `qel:s(x,p,t)` is known to be equal to '`dc:title`'. This enables SWARD to drastically reduce the number of clauses in the disjunction shown in Example 2 using compile time evaluation of the `SM` table. Once `SM` is evaluated there is only a single clause left (see Example 3) from the original disjunction which is translated to a single SQL expression. An interesting situation arises when `p` is unknown. In SQL the column names of a query must be explicitly specified. This is critical for relational database query optimization. By contrast RDF queries can have dynamic properties, i.e. RDF queries can contain variables bound to RDF predicates. This would correspond to variables bound to table columns in SQL, which is not allowed. For QEL this means that the calculus predicates such as `qel:s` may be constructed out of variables rather than of known RDF resources.

A *statement cache* is used in SWARD to recycle already compiled QEL queries meaning that the query can be executed directly without relational query optimization.

Table 2. Measuring execution time of *q2* over variable sized table `Resource` without compile time evaluation

Tuples	Proc	s_{Proc}	Exec	s_{Exex}
700	8.667	0.013	0.005	0.008
2000	8.802	0.015	0.010	0.009
5000	8.786	0.027	0.020	0.009
10000	8.776	0.023	0.016	0.007

To see how compile time evaluation can improve performance an experiment was conducted on a PC with a Pentium 3 2.2 GHz processor with 512 RAM running *Microsoft SQL server*. A QEL query was executed repeatedly over the relational database *WM* with one table, `Resource`. The table was scaled up in each test with 700, 2000, 5000, and 10000 tuples. For every table size the test was made first without, see Table II, and then with compile time evaluation, see Table III. The query used in the experiment, *q2*, is an extension of the query in our running example and is expressed in QEL as:

```
@prefix qel:http://www.edutella.org/qel#
@prefix dc:http://purl.org/dc/elements/1.1/
?(x,t,d):-qel:s(x,'dc:title',t),
          qel:s(x,'dc:description',d),
          qel:like(t,'Matter'),
          qel:like(d,'Fysik')
```

In natural language that would be: Find all museum artifacts that have a name that contains the string '`Matter`' and a description that contains the string '`Fysik`'[5].

The time for each test was measured as the time taken to process the query; **Proc** and the time taken to execute the query; **Exec** as part of **Proc**. For the purpose of our experiment and for simplicity, **Exec** was defined as the time spent calling SQL. All tests were measured in seconds. Table II and Table III respectively show the results of the experiment with or without compile time evaluation. For each test the mean value was calculated and chosen as the result of the test. Standard deviations for **Proc** (s_{Proc}) and **Exec** (s_{Exec}) were also calculated.

Table 3. Measuring execution time of *q2* over variable sized table `Resource` with compile time evaluation

Tuples	Proc	s_{Proc}	Exec	s_{Exec}
700	0.953	0.017	0.010	0.009
2000	1.130	0.032	0.020	0.005
5000	1.140	0.027	0.020	0.006
10000	1.115	0.025	0.020	0.006

When analyzing the result of the experiment we see that there is a noticeable reduction in time spent processing *q2*, **Proc**, when using compile time evaluation

[5] English:'Physics'.

compared to when not. However, the execution time, **Exec**, is more or less constant. The experiment shows that query processing time is improved substantially by compile time evaluation of the SM table. However, the query execution time **Exec** is not affected by compile time evaluation. The reason is that the query optimizer in SWARD knows that `SM` is local to the wrapper. Therefore it is evaluated before accessing the more expensive relational back-end and this makes query execution efficient.

7 Summary

The SWARD system provides RDF views over relational databases in terms of *virtual* statements. Transformations according to some general translation rules yield optimized RDF queries in terms of domain calculus expressions. An algebra generator then produces a query plan out of these expressions that contains calls to functions executing SQL queries.

Various techniques are used to optimize the expressions such as rewrites and push down of filter operators (e.g. `like`) to the relational database. This allows for the creation of RDF views over relational databases in a highly scalable way.

When translating semantic web queries to SQL there is a problem that, unlike SQL, semantic web queries are dynamic and they are not necessarily expressed in terms of a schema. However, SWARDs abstraction of relational database columns into CSSs makes the translation of QEL queries into SQL natural. To execute a QEL query, q, over an RDF view w means evaluating $w \wedge q$ as illustrated in Example 2. To retrieve the entire view means executing all these expressions and appending their results. Thus RDF predicates become large disjunctions of clauses where each clause is a conjunction of constraints in the QEL query and calls to the relational database. Since the disjunctions are large it is important to reduce their size before generating the query algebra expression. We have shown that compile time evaluation of the source mappings between URIs of different terminologies allows for substantial reduction of the calculus expression and improved query compilation time. However, in our example the query execution time is not effected since the regular cost-based query optimization produces an efficient execution plan in any case.

We are currently generalizing our approach to include more complex queries and other optimization strategies.

SWARD provides views over relational databases only in terms of basic RDF data. Future work will offer a semantically enriched RDFS form. This requires creating for each statement source some additional virtual statements providing information about class-subclass relationship, instance-of etc.

References

1. Barrett et al.: RDF Representation of Metadata for Semantic Integration of Corporate Information Resources, *Proc. WWW2002*, 2002.

2. D.Beckett and J.Grant: SWAD-Europe: Mapping Semantic Web Data with RDBMSes, *http://www.w3.org/2001/sw/Europe/reports/scalable_rdbms_mapping_report/*, 2001.
3. T.Berners-Lee, J.Hendler, and O.Lassila: The Semantic Web, *Scientific American*, May 2001.
4. D.Brickley and R.V.Guha: RDF Vocabulary Description Language 1.0: RDF-Schema, *http://www.w3.org/TR/2004/REC-rdf-schema-20040210/*, 2004.
5. C.Bizer: D2R MAP - A Database to RDF Mapping Language, *The 12th International World Wide Web Conference (WWW2003)*, Budapest, Hungary, 2003.
6. V.Christophides, G.Karvounarakis, A.Magkanaraki, D.Plexousakis, and V.Tannen: The ICS-FORTH Semantic Web Integration Middleware (SWIM), *Data Engineering Bulletine*, IEEE, 26(4), Dec. 2003.
7. S.Decker et al.: The Semantic Web - on the Roles of XML and RDF, *IEEE Internet Computing*, Sept./Oct. 2000.
8. Dublin Core Meta-data Initiative, Dublin Core Metadata Element Set, V 1.1, *http://dublincore.org/documents/dces/*
9. G. Fahl and T. Risch: Query Processing over Object Views of Relational Data, *The VLDB Journal*, Springer, Vol. 6, No. 4, 261-281, 1997.
10. H. Garcia-Molina et al.: The TSIMMIS Approach to Mediation: Data Models and Languages, *Intelligent Information Systems (JIIS)*, Kluwer, 8(2), 117-132, 1997.
11. L. Haas, D. Kossmann, E.L. Wimmers, and J. Yang: Optimizing Queries across Diverse Data Sources, *Proc. 23rd Intl. Conf. on Very Large Databases (VLDB'97)*, 276-285, 1997.
12. V. Josifovski and T. Risch: Integrating Heterogeneous Overlapping Databases through Object-Oriented Transformations, *Proc. 25th Conference on Very Large Databases (VLDB'99)*, 435-446, 1999.
13. G.Karvounarakis el al.: Querying the Semantic Web with RQL, *Computer Networks and ISDN Systems Journal*, 42(5), 617-640, August 2003.
14. G.Klyne and J.J.Carroll: Resource Description Framework (RDF): Concepts and Abstract Syntax, *http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/*, 2003.
15. L.Liu and C.Pu: An Adaptive Object-Oriented Approach to Integration and Access of Heterogeneous Information Sources, *Distributed and Parallel Databases*, Kluwer, 5(2), 167-205, 1997.
16. A.Magkanaraki, V.Tannen, V.Christophides, and D.Plexousakis: Viewing the Semantic Web Through RVL Lenses, *2nd International Semantic Web Conference (ISWC'03)*, Sanibel Island, Florida, USA, 2003.
17. *http://www.museifonstret.se/*
18. W.Neidl et al.: EDUTELLA: A P2P Networking Infrastructure Based on RDF. *Proc. 11th International World Wide Web Conference*, Honolulu, Hawaii, USA, 2002.
19. D. Quass, A. Rajaraman, Y. Sagiv, J.Ullman, and J. Widom: Querying Semistructured Heterogeneous Information in Deductive and Object-Oriented Databases, *Proc. of the DOOD'95 conference*, LNCS Vol. 1013, 319-344, Springer 1995.
20. RDF Gateway - a platform for the semantic web, Intellidimension, *http://www.intellidimension.com/*.
21. RDF Query Exchange Language (QEL) - concepts, semantics and RDF syntax, *http://edutella.jxta.org/spec/qel.html*
22. T.Risch, V.Josifovski, and T.Katchaounov: Functional Data Integration in a Distributed Mediator System, in P.Gray, L.Kerschberg, P.King, and A.Poulovassilis (eds.): *Functional Approach to Computing with Data*, Springer, 2003.
23. A.Seaborne: RDQL - A Query Language for RDF, W3C Member Submission, *http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/*, 2004.
24. DD.Straube, and MT.Özsu: Queries and query processing in object oriented database systems, *ACM Transaction Information Syst*, 8(4), 1990.

25. A. Tomasic, L. Raschid, and P. Valduriez: Scaling Access to Heterogeneous Data Sources with DISCO, *IEEE Transactions on Knowledge and Date Engineering*, 10(5), 808-823, 1998.
26. K.Wilkinson, C.Sayers, H.A.Kuno, and D.Reynolds: Efficient RDF storage and retrieval in Jena 2, *1st Intl. Workshop on Semantic Web and Databases (SWDB'03)*, Berlin, http://hplabs.hp.com/techreports/2003/HPL-2003-266.html, 2003.
27. J. Ullman: *Database and Knowledge Base Systems*, Vols 1 & 2, Computer Press, 1988.

Inverse Wrappers for Legacy Information Systems Migration

Jean Henrard[1 2], Anthony Cleve[1], Jean-Luc Hainaut[1]

[1] Database Applications Engineering Laboratory
Institut d'Informatique, University of Namur
rue Grandgagnage, 21 - B-5000 Namur - Belgium
{jhe, acl, jlh}@info.fundp.ac.be

[2] REVER S.A.
Boulevard Tirou, 130 - B-6000 Charleroi - Belgium

Abstract. The paper studies some problems that arise when a technology change induces the migration of a data-centered application. In particular, it addresses the difficult problem of migrating application programs from a legacy data manager, such as a COBOL file system, to a modern DBMS, such as a relational database management system. The approach suggested in this paper relies on the concept of inverse wrappers, that is, wrappers that simulate the legacy API on top of the new database. This architecture allows (1) the design of a fully normalized database rid of the anomalies of the legacy data, (2) future programs to be developed on a sound basis and (3) legacy programs to work on the new database with minimum transformation, and therefore at low cost.
The paper describes the components of this architecture, a methodology to design them and a CASE tool that automates their generation.

1 Introduction

Migrating large information systems has long been recognized to be one of the most complex and failure-prone processes. Several migration strategies have been identified and described in the literature, notably in [3]. They can be classified according to several dimensions.

Identifying two major components, namely the data and the programs, we can distinguish two families of strategies, according to which component is migrated first.

- *Database first strategies*. First, the legacy database is migrated, so that new programs can be developed on the target platform; later on, the legacy programs are migrated to the new database; in the mean time, they either keep using the legacy database, which is synchronized with the new database, or they access the latter through some sort of wrappers.
- *Database last strategies*. First, the programs are migrated to the new platform; they then use the legacy database through wrappers. New applications access the legacy database through the same interface. When all the applications have been converted, the database itself is migrated.

The second dimension concerns the time frame within which the replacement is carried out. One typically identifies two main families.

- *Big bang approach*. The new system, comprising the data and the programs, replaces the legacy system in one step. Most generally, the substitution is carried out in a very short time, typically a few days, so that both systems run with no overlap.
- *Chicken little approach*. The database and the applications are migrated piece by piece.

This paper addresses the database first strategy in which the new database completely replaces the legacy one, so that their lifespan do not overlap. More specifically, it discusses the problem of migrating the legacy programs through a chicken little approach in a reliable and inexpensive way through the use of wrappers.

In migration and interoperability architectures, wrappers are popular components that convert legacy interfaces into modern ones. For instance, a set of standard files is given an object-oriented API suited to modern distributed architectures. Such wrappers are used to renovate legacy components. The components discussed in this paper play the inverse role: they provide access through a legacy API to the new database. Hence the name of inverse wrapper.

We will show how to build the wrapper for the database first strategy. This strategy involves the initial migration of the legacy database to a modern one and then (incrementally) migrates the legacy applications and interfaces. The advantage of this solution is that the data are migrated to new structures that are supposed to be well designed and rid of the flaws and awkwadness of the legacy database. The legacy applications can still be used thanks to a wrapper that simulates the access to the legacy database on top of the new one. On the contrary, the new applications as well as the migrated ones can directly access the new database and profit from its expressiveness.

The reminder of the paper is organized as follows. Section 2 presents our migration strategies. Section 3 develops how the schema of the legacy database is reengineered. Section 4 presents how the mapping between the legacy database schema and the new one can be modeled. Section 5 shows how the data are migrated. Section 6 describes the wrapper generation. Section 7 presents how the programs are transformed to invoke the wrapper instead of accessing the legacy database. Section 8 describes some practical aspects of the wrapper generation and section 9 concludes this paper.

2 Problem Statement

One of the biggest challenges in database migration based on a technology change, is to provide new data structures that translate all the semantics of the legacy database and nothing else. In particular, these structures should not inherit any technology-dependent feature from the legacy data structures.

Failing to meet this requirement would provide a database that is flawed from its very start, and that will lead to increasing semantic, integrity and performance problems.

The most popular migration approach, that could be called one-to-one strategy, ensures the structural equivalence between both legacy and new data structures. In a file to relational database migration, it consists of converting each record type into a table and each top-level field into a column. Its popularity comes from its extreme simplicity

and its low cost : both databases have (as far as possible) the same schema and converting the programs is particularly easy, since each I/O statement is replaced with a functionally equivalent DML[1] sequence. No understanding of the data structures nor of the processing logic is required, so that the translation process can be automated to a large extent. Such an approach naturally yields, but in some exceptional situations, poor data structures that are difficult to maintain and to evolve.

The other approach, that we can call semantics-based strategy, has been advocated by the authors in former papers [4]. It consists of converting the legacy database into new, normalized, data structures that ensure semantic equivalence only. For example, a record type can be translated into several normalized tables, while several record types could be merged into a single table. The new database is stricty independent from the legacy technology and no longer suffers from the flaws and idiosyncrasies of the legacy database.

Unfortunately, this strategy can be much more costly than the former one. Indeed, it requires a deep understanding of the legacy data structures before translating them into the target data model. Secondly, since both data structures generally are quite different, data conversion involves complex data transformations that go well beyond the straightforward store-each-record-into-a-row technique of the former approach. Thirdly, the conversion of the programs is more complex since a legacy I/O statement could have to be developed into a complex procedure the writing of which may require an in-depth understanding of the application logic.

The goal of this paper is to explore and develop solutions to the problems raised by the semantics-based approach.

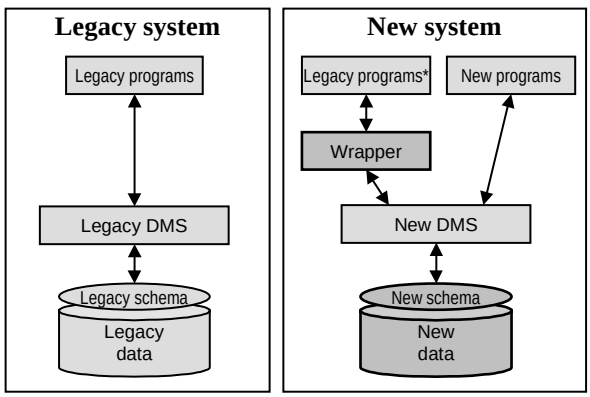

Fig. 1. System conversion

The migration strategy we describe in this paper is sketched in Fig.Fig. 1. The left part shows the main parts of the legacy system, comprising programs that interact with the legacy data through its legacy schema.

1. Data Manipulation Language

The right part shows the state of the new system after the legacy DMS[1] has been replaced with a modern DMS (New DMS). The new database comprises the migrated data and the migrated schema. Legacy programs now access the data through an inverse wrapper that simulates the API of the legacy DMS. These programs have been slightly processed in order to syntactically comply with the wrapper programming interface. New programs use the database through the native interface of the new DMS. Later on, if and when needed, the legacy programs could be rewritten according to the new technology.

This discussion shows clearly the new components that have to be produced in the migration process: the new database schema, the new data, the wrapper and the transformed programs. A fifth component will be mentioned later on.

We will show that they can be automatically derived from the reengineering of the database schema.

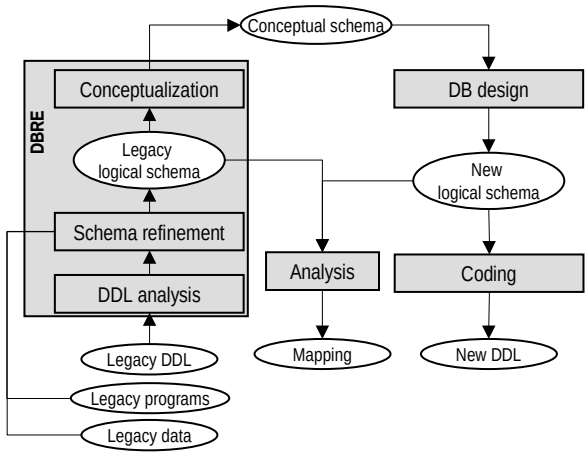

Fig. 2. Database Schema Reengineering

This process is known to include two main phases, namely Database reverse engineering and Database design (Fig. 2). The former aims at recovering the semantics of the legacy data structures, that is, their conceptual schema, from various sources such as the DDL[2] code, the source code of the programs and the data themselves, while the latter implements this schema into a physical schema, in the form of DDL statements for the new DMS (New DDL).

Since the reengineering process is supported by a CASE tool, the mapping between the legacy schema and the new schema can be recorded. Therefore, this process yields an important by-product, the Mapping.

Converting the new data involves some data extract/transform/load (ETL) component that we will call Migrator for short (Fig. 3). This component is the fifth product of the migration methodology.

1. Data Management System
2. Data Description Language

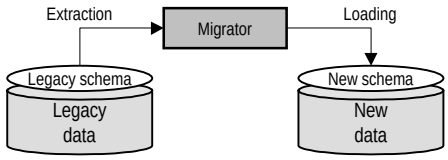

Fig. 3. Data conversion

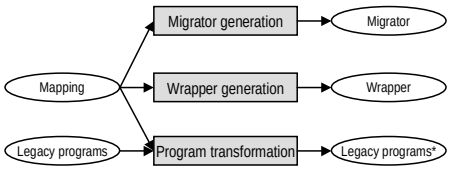

Fig. 4. Production of the last components of the new system

We now have to show how to produce the data migrator and the wrapper, and how to transform the legacy programs. They are built through three additional engineering processes that are illustrated in Fig. 4.

In the next section, we develop further the processes that we have put in light in this discussion. In particular, we will describe the functions of the DB-MAIN[1] CASE tool that support each of them.

3 Schema Reengineering

The schema reengineering, or conversion, process analyzes the legacy applications to extract the logical and conceptual schemas of the legacy database through a database reverse engineering (DBRE) phase. Then this conceptual schema is transformed into the logical schema of the new database through a classical database design process, then is coded into the DDL of the new DMS.

3.1 Methodology

Fig. 2/left depicts a simplified version of the methodology used to perform DBRE. A complete presentation of this methodology can be found in [5]. The DDL analysis parses the legacy DDL code to retrieve the raw physical schema. In the schema refinement process, the schema is refined through an in-depth inspection of the way the program uses and manages the data. Through this process, additional structures and constraints are identified, which were not explicitly declared but expressed in the procedural code. The existing data can also be analyzed, either to detect constraints, or to confirm or discard hypotheses on the existence of such constraints. The final DBRE step is the data structure conceptualization interpreting the legacy logical schema into the conceptual schema. Both schemas have the same semantics, but the latter is platform independent and includes no technical constructs.

1. www.db-main.be

The logical schema of the new database is derived from the conceptual schema through standard database engineering techniques (Fig. 2/right). This schema is then enriched with technical constructs specific to the new platform and is then used to generate the DDL code of the new database.

4 Tool Support

Extracting the raw physical schema and storing it in the CASE tool repository is done through a DDL extractor (SQL, COBOL, IMS, CODASYL, RPG, etc.) from the parser library.

Schema refinement requires powerful and customizable schema and program analyzers, such as program slicing, pattern matching, etc. Experience showed us that there is no such thing as two similar reengineering projects. Hence the need for programmable, extensible and customizable tools. DB-MAIN (and more specifically its meta functions) includes features to extend its repository and its functions. In particular, it includes a 4GL (Voyager2) that allows analysts to develop their own customized processors [5].

Data structure conceptualization and database design are based on schema transformations, that will be discussed below. Code generators produce the DDL code of the new database according to the specifications of logical schema.

5 Mapping Definition

Deriving a schema from another is performed through techniques such as renaming, translating, conceptualizing, which basically are schema transformations. Most database engineering processes can be formalized as chains of schema transformations as demonstrated in [6].

5.1 Schema Transformation

Roughly speaking, a schema transformation consists of deriving a target schema *S'* from a source schema *S* by replacing construct *C* (possibly empty) in *S* with a new construct *C'* (possibly empty). Adding an attribute to an entity type, replacing a relationship type with an equivalent entity type or with a foreign key are three examples of schema transformations.

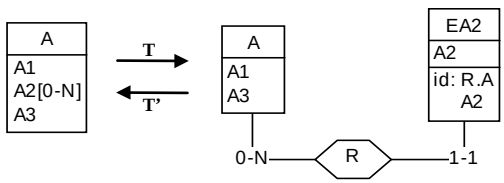

Fig. 5. Representation of the structural mapping of a transformation that replaces multivalued attribute A2 with entity type EA2 and relationship type R

More formally, a transformation Σ is defined as a couple of mappings <*T,t*> such as: *C' = T(C)* and *c' = t(c)*, where *c* is any instance of *C* and *c'* the corresponding instance of *C'*.

Structural mapping *T* explains how to modify the schema while instance mapping *t* states how to compute the instance set of *C'* from the instances of *C*.

Any transformation Σ can be given an inverse transformation Σ' = <*T',t'*> such that *T'(T(C))=C*. If, in addition, we also have: *t'(t(c))=c*, then Σ and Σ' are said semantics-preserving (Fig. 5).

5.2 Compound Transformation

A compound transformation *T = T2 o T1* is obtained by applying *T2* on the schema that results from the application of *T1* [6].

An important conclusion of the transformation-based analysis of database engineering processes is that most of them, including reverse engineering and database design, can be modeled through semantics-preserving transformations. For instance, transforming a conceptual schema *CS* into a logical schema *LS* can be modeled as a compound semantics-preserving transformation *C-to-L = <CS-to-LS, cd-to-ld>* in such a way that *LS = CS-to-LS(CS)*. This transformation has an inverse: *L-to-C = <LS-to-CS, ld-to-cd>* such as *CS = L-to-C(LS)*. The latter is a formal model of the Conceptualization phase of DBRE (Fig. 2).

5.3 Transformation History

The history of a database engineering process consists of the formal trace of its chain of transformations. In a history, a transformation is entirely specified by its signature, which specifies the name of the transformation, the name of the objects concerned in the source schema and the name of the new objects in the target schema. More precisely, the history of a compound transformation contains the signatures of each transformation according to their order in the chain. For example, the signatures of the inverse transformations represented in Fig. 5 are:

```
T :(EA2,R) ← ATTRIBUTE-to-ET/inst(A,A2)
T':(A2) ← ET-to-ATTRIBUTE(EA2)
```

The first expression can be read as follows: by application of the *ATTRIBUTE-to-ET/inst* transformation on attribute *A2* of entity type A, a new entity type *EA2* and a new relationship type *R* are created. To simplify, certain objects implied in the transformation are not specified in the signature. This is the case for the relationship type *R* which disappears when applying *ET-to-ATTRIBUTE*.

5.4 Source and Target Logical Mappings

The mappings between the source and target logical schemas are modeled through a transformation history. The history is defined by the trace of the complex compound transformation: *LegLS-to-NewLS = LS-to-CS o CS-to-LS* in such a way that *NewLS = CS-to-LS(LS-to-CS(LegLS))*, where *NewLS* is the new logical schema and LegLS is the legacy logical schema.

5.5 Support

DB-MAIN includes a rich toolkit of transformations, most of them being semantics-preserving. In addition, it can record the history of the transformations applied to convert the legacy logical schema into a conceptual schema and to transform the latter into the new logical schema.

6 Data Conversion

Schema conversion is concerned with the conversion of the data format and not of its content [1]. Data conversion is taken in charge by a software component often called ETL processor (Fig. 3), which transforms the data from the data source to the format defined by the target schema. A converter has three main functions. Firstly, it performs the extraction of the data source. Then, it converts these data in such a way that their structures match the target (new) format. Finally, it writes legacy data in the target format. A converter relies on the mappings between the source and target physical schemas.

The analysis of the Schema reengineering process has shown that, through schema refinement, implicit constraints can be discovered, that will be translated into the new schema. Quite often, data may violate these constraints, so that the legacy data often have to be cleaned before being migrated. Though data cleaning can be a complex task, it can be partly automated through specific functions of the migrator.

6.1 Methodology

Data conversion involves three main tasks. Firstly, the legacy logical schema is converted into the new logical schema. Secondly, the mapping between the source and target physical schemas is extracted. Finally, this mapping is implemented in the migrator for translating the legacy data according to the format defined in new logical schema.

6.2 Tool Support

Writing data migrators manually is an expensive and complex task, particularly for mapping as we find them in semantics-based migration. As told in section 4.5, DB-MAIN offers the mechanisms to record transformation histories. It also provide a translator of histories into mapping between the legacy and new schemas. Several migrator generators have been developed for various technology conversions. Such a generator automatically derives the migrator code from the mapping between both logical schemas.

It must be noted that full automation generally is not achieved for cost reason. Indeed, some mappings are problem-specific and have not been coded in the generator. In such cases, the analyst has to add the specific code in the migrator. For example, if some specific data conversion is needed (convert measuring units, capitalize a string, add a prefix to a field value, etc.) the user must write the conversion code and insert it in the migrator.

In order to design a more general solution and to minimize hand-written migrator generator sections, we have divided the data migrator in two modules. The first one

reads the source data and produces an XML file. The second reads the XML file and stores the data into the target database.

The structure of the XML file (DTD) is intermediate between the source schema and the target schema, in order to ease the migration. The only purpose of the XML document is to facilitate the transfer of data. Thus we did not try to have an XML file that express the semantics of the data, as if it had to be used to store and manipulate the data. For example, if the data contains information about customers and orders, we do not necessarily produce a file where the orders are encapsulated into the data of its customer. Usually the structure of the XML file is very close to the structure of the legacy schema.

7 Wrapper Generation

A data wrapper is a data model conversion component that is called by the application program to carry out operations on the database. Its goal is generally to provide application programs with a modern interface to a legacy database (e.g., allowing Java programs to access COBOL files). In the present context, the wrapper is to transform the renovated data into the legacy format, e.g., to allow COBOL programs to read, write records that are built from rows extracted from a relational database.

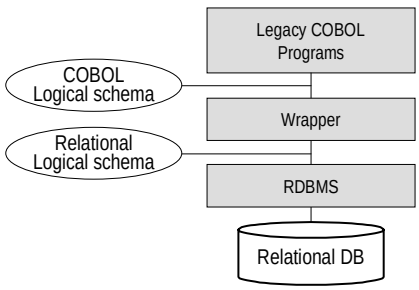

Fig. 6. A wrapper allows the new database to be accessed by the legacy application

The wrapper converts all legacy DMS requests from legacy applications into requests against the new DMS that now manages the data. Conversely, it captures results from the new DMS, possibly converts them to the appropriate legacy format [7] and delivers them to the application program. Fig. 6 depicts a frequent pattern, in which the legacy logical schema relying on the COBOL file interface is rebuilt from the new relational logical schema.

7.1 Methodology

As part of the InterDB project [9], we have developed a technology for the automated production of hard-coded wrappers for data systems. In [8], we have shown that the code of a wrapper dedicated to a couple of DMS families, performs the structural and instance mappings and that these mappings can be modeled through semantics-preserving transformations. For a consultation query, the structural mapping explains how to translate the query while the instance mapping explains how to form the result instances. Consequently the *LegLS-to-NewLS* mapping provides the necessary and sufficient information to produce the procedural code of a specific wrapper.

7.2 Support

As for data conversion, writing wrapper manually is an expensive and complex task. Thanks to the DB-MAIN representation of the mapping between the two schemas, it is possible to automatically generate the wrapper.

The main difficulty is that the wrapper must simulate the legacy DML request from the legacy applications into requests against the new DMS. For example, the COBOL start instruction must be translated in several SQL queries. Section 8 will give some examples of such conversion rules.

Until now, we have implemented wrappers for COBOL-to-SQL and for IDS/II-to-SQL. Our approach consists of defining one wrapper for each record type of the legacy database.

For the COBOL-to-SQL wrapper, this solution is obvious since there is no relation between the record types. Through the analysis of the DML instruction it is possible to figure out which record is manipulated and thus which wrapper needs to be called.

For the IDS/II-to-SQL, the solution of one wrapper per record type is more less obvious. Its main advantages is that the wrappers are smaller and easier to generate and to maintain. But some DML instructions are more difficult to translate. For example, the IDS/II DML get statement does not specify the record type to be read since this type is that of the last record type accessed. If we use one wrapper per record type, the test figuring out which was the last record type accessed must be done in the legacy application and not in the wrapper, an option that makes program transformation more difficult.

8 Program Transformation

The database first migration strategy attempts, in a first step, to keep the legacy program logic unchanged and to map it on the new DMS technology [3]. In the previous section, we have explained that it is possible to build wrappers that transform the renovated data into the legacy format. In this way, the application programs invoke the wrapper instead of the legacy DMS. If the wrapper simulates the modeling paradigm of the legacy database and its interface, the alteration of the legacy code is minimal. It consists of replacing the DML statements with wrapper invocations.

Some legacy DMS, such as MicroFocus COBOL, provide an elegant way to interface wrappers with the legacy code. They allow programmers to replace the standard DMS library with customized library. In this case, the legacy code does not need to be altered at all.

8.1 Methodology

Thanks to the use of the wrapper-based technique, the logic of the legacy programs does not need to be modified. Some program transformations are performed to replace each DML instruction with a call to the wrapper. The program transformation is quite simple because the analysis of the DML instruction is sufficient to derive the parameters of the wrapper call.

8.2 Support

To automate the program transformations, we rely on the ASF+SDF meta-environment developed by the SEN1 research group of the CWI [10]. We define the grammar of the legacy language and the transformation rules to produce our program transformation tool [11]. The tool takes the legacy program and some extra parameters (such as the name of the wrapper) to produce the renovated program.

9 Practical Aspects

9.1 Partial Migration

In many projects, not all the database need to be migrated. For example, the legacy applications may access data of another information system that is not migrated. To cope with such a situation, only the DML instructions used to access the migrated database are translated by a call to the wrapper and the other DML instructions, that access the non migrated data, are left unchanged.

9.2 Model Conversion Restrictions

Some legacy DMS constructions and some DML instructions cannot be easily translated into the new DMS or implemented into the wrapper. The programs that use such constructs need to be modified or redeveloped. This is especially true for the physical constructs that are not implemented in the new DMS, but that are used by the application programs. We briefly mention some of those we have met during the development of the IDS/II-to-SQL wrapper:

- **`DB-KEY`**. In IDS/II it is possible to get the current database key, i.e., some kind of pointer to the current record. This `DB-KEY` can be stored in a variable or in a database field (though not recommended) and used later to access the record (`FIND DB-KEY IS <var-name>`). The SQL `rowid` is no substitute since its scope is the current table. If the `DB-KEY` is only used locally in a program it can be simulated. However, its use as a foreign key requires complex artifacts such as technical table and columns. In this specific project, we have decided to not translate the `DB-KEY` construct and thus to rewrite the parts of the programs that use it.
- **`Find within area`**. In the declaration of the IDS/II database, each record type is assigned to one or several areas (files). It is possible to walk through all the records of an area (independently of their type): `find first/next within <area-name>`. This is a very efficient way to access sequentially all the records of an area. But in the new (SQL) database, the areas are not necessarily translated into `db_spaces`. In addition, SQL provides no way to scan the rows stored in a `db_space`. If each record type is only stored in one area, this can be translated by the access of each record type of the area one after the other. We have chosen to not offer this kind of access and to rewrite those parts of the legacy programs.

9.3 Illustration

To illustrate the building of wrappers, we will show how COBOL `START` and `READ NEXT` statements have been translated.

The change of paradigm when moving from COBOL files to relational database induces currency management problems such as the identification of the sequence scan. COBOL allows the programmer to start a sequence based on an indexed key (START statement), then to go on in this sequence through READ NEXT statements. The most obvious SQL translation is through a cursor-based loop. However, the READ NEXT statements can be scattered throughout the program, so that the identification of the initial START statement, specifying the current sequence scan, is complex.

The technique illustrated below solves this problem. Let STOCK be an indexed COBOL file such as declared in Fig. 7/left. The record type STK can be translated into the SQL table STOCK (see Fig. 7/right).

```
SELECT STOCK ASSIGN TO "STOCK"
   ORGANIZATION IS INDEXED
   ACCESS MODE IS DYNAMIC
   RECORD KEY IS STK-CODE.
 ...
FD STOCK.
  01 STK.
    02 STK-CODE PIC 9(5).
    02 STK-NAME PIC X(100).
    02 STK-LEVEL PIC 9(5).
```

```
create table STOCK (
   CODE numeric(5) not null,
   NAME char(100) not null,
   LEVEL numeric(5) not null,
   primary key (CODE))
```

Fig. 7. STOCK file declaration and its migrated table

A SQL cursor is declared for each kind of record key usage (=, >, >=) in the wrapper. For instance, the table STOCK gives three cursors. Among them:

```
DECLARE G_STK_CODE CURSOR FOR
  SELECT CODE, NAME, LEVEL
  FROM STOCK
  WHERE CODE > :WR-STK-CODE
  ORDER BY CODE
```

```
* translation of "START STOCK"
  IF WR-OPTION = "KEY IS > STK-CODE"
    EXEC SQL
      SELECT COUNT(*)
      INTO :WR-COUNTER
      FROM STOCK
      WHERE CODE > :WR-STK-CODE
    END-EXEC
    IF SQLCODE NOT = 0
      SET WR-STATUS-INVALID-KEY TO TRUE
    ELSE
     IF WR-COUNTER = 0
     SET WR-STATUS-INVALID-KEY TO TRUE
     ELSE
      MOVE STK-CODE TO WR-STK-CODE
      EXEC SQL
        OPEN G_STK_CODE
      END-EXEC
     MOVE "G_STK_CODE" TO LAST-CURSOR.
    ...
```

```
* translation of "READ STOCK NEXT"
  IF LAST-CURSOR = "G_STK_CODE"
    EXEC SQL
      FETCH G_STK_CODE
      INTO :WR-STK-CODE,
           :WR-STK-NAME,
           :WR-STK-LEVEL
    END-EXEC
    IF (SQLCODE = 0)
      MOVE WR-STK-CODE TO STK-CODE
      MOVE WR-STK-NAME TO STK-NAME
      MOVE WR-STK-LEVEL TO STK-LEVEL
    ELSE
      SET WR-STATUS-AT-END TO TRUE.
  ...
```

Fig. 8. Translation of START STOCK and of READ STOCK NEXT

A START statement can be simply translated through the corresponding cursor opening (Fig. 8/left). Note that wrappers also have to simulate the legacy DMS exceptions, that are taken into account in the legacy programs logic. For instance, the execution of the COBOL START statement can produce an INVALID KEY exception occur-

ring when no record with the specified key value has been found. In the example above, the wrapper returns the variable `WR-STATUS` simulating the COBOL input-ouput exceptions.

The SQL translation of the `READ NEXT` statement consists of a `FETCH` of the last opened cursor, as shown in the example of Fig. 8/right

The variable `LAST-CURSOR` provides the name of the currently opened cursor on the table `STOCK`. So we know which cursor has to be fetched with respect to the current reading sequence.

10 Conclusion

Coupling two independent technologies, namely schema transformation and program transformation, provides us with a very powerful tool to help solve the problem of automatically generating the components of semantics-based system migration.

Practically, several experiments conducted for two years have shown the validity of the approach, at least for small to medium size programs. Applying it to large scale systems is planned in the near future.

As discussed in the paper, the prototype CASE tool still is experimental. Beyond the proof of concept that is described in the paper, improvements are being studied in three directions.

1. Refining the mappings for the COBOL-to-SQL and IDS/II-to-SQL migrations, in particular, by extending the database with technical temporary structures that simulate dbkeys and areas during the transition period.
2. Collecting performance figures for large scale system and, where needed, designing more efficient wrapper strategies.
3. Developing mapping rules for two other legacy models, i.e., IMS and SQL. Indeed, SQL-to-SQL and IMS-to-SQL migrations are economically important but require specific rules.

11 References

[1] Aiken, P., Muntz, A., Richards, R.: "DOD Legacy Systems - Reverse-Engineering Data Requirements", Communications of the ACM, May 1994.

[2] Bull: DSE I-D-S/II (COBOL) Reference Manual, Bull, 1993.

[3] Brodie, M. L., Stonebraker, M.: Migrating Legacy Systems: Gateway, Interfaces & the Incremental Approach, Morgan Kaufmann, 1995.

[4] Henrard, J., Hick, J-M., Thiran, Ph., Hainaut, J-L.: "Strategies for Data Reengineering", in Proeedings. of WCRE'02, IEEE Computer Society Press, 2002.

[5] Hainaut, J.-L., Roland, D., Hick, J.-M., Henrard, J., Englebert, V.: "Database Reverse Engineering: from Requirements to CARE Tools", Journal of Automated Software Engineering, 3(1), 1996.

[6] Hainaut, J.-L.: "Specification preservation in schema transformations - Application to semantics and statistics", Data & Knowledge Engineering, 16(1), Elsevier Science Publish, 1996.

[7] Papakonstantinou, Y., Gupta, A., Garcia-Molina, H., Ullman, J.: "A Query Translation Scheme for Rapid Implementation of Wrappers", in Proceedings of the international Conference on Declarative and Object-oriented Databases, 1995.

[8] Thiran, Ph., Hainaut, J.-L.: "Wrapper Development for Legacy Data Reuse", in Proceedings of WCRE'2001, IEEE Computer Society Press, 2001.

[9] Thiran, Ph., Hainaut, J-L., Integration of Legacy and Heterogeneous Databases, LIBD Publish., Namur, 2002.

[10] van den Brand, M.G.J., Klint, P. : ASF+SDF Meta-Environment User Manual, 2003.

[11] Cleve, A. : Data-centered Applications Conversion using Program Transformations, Master's Thesis, University of Namur, 2004.

Developing robust wrapper-systems with Content Based Recognition

Christoph Göldner and Thomas Kabisch and Jörn Guy Süß
{goeldner,tkabisch,jgsuess}@cs.tu-berlin.de

Technical University of Berlin
Computation and Information Structures (CIS)

Abstract. Maintenance is a limiting factor in the application of wrapper technology. This paper introduces an approach which minimizes maintenance through simplified definition and automatic adaptation. It reuses WrapIt[12] as an existing component to accelerate initial definition and user-oriented maintenance, and exploits master-detail relations often found in web applications to make the wrapper resistent to cosmetic changes. Changes are detected based on structural information and corresponding sample data.

1 Introduction

Wrapper systems can provide structured access to the content of the hidden web[13]. But their application is limited by the effort and cost of initial definition, lifetime maintenance and response time issues. While initial definition has been extensively treated in research[6, 3] the other two aspects play a more limited role. This paper introduces a simple struture-based approach, which specifically addresses the last two aspects, while treating the first as a variable component. As an area of application, it is intended to be applied by lay internet users. The approach targets the bulk of applications working on a fixed schema of form-based query, master view and subsequent detail view.

The rest of the paper is structured as follows: Section 2 gives an overview of the approach through an example. Section 3 formalizes the approach based on relational theory and describes the related algorithms in detail. Section 4 presents related research. The paper closes with an outlook in section 5.

2 System Overview

Based on a real world example, the Internet Movie Database, we introduce our wrapping approach. The Internet Movie Database http://www.imdb.com/ is a database driven search site which follows an archetypical navigation structure: An input form allows the user to formulate a query, a master page subsequently

displays the first set of results in a list-like structure. Figure 2 shows an excerpt of such a master page. Subsequent master pages are appended if the number of results exceeds a certain limit. From the master page, details are accessible by selection of a link. The detail page reflects the key information visible on the master page and additional detail information. Master and detail pages follow a fixed structure. Figure 1 describes master/detail page scheme.

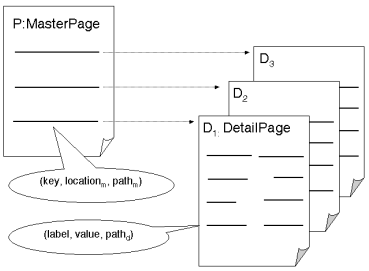

Fig. 1. Informal model of dynamically generated master/detail page scheme

The following sections show how our approach defines, queries and reconfigures wrappers for web sources using this design scheme.

2.1 Initial Definition

As mentioned in the introduction, there are a number of approaches to simplify the initial defintion of the wrapper and the relationships between query, master and detail. In our approach, we use WrapIt[12], to identify candidates for the varying elements of the page which the lay user only has to reject or confirm. Further we use XWrap [10] to extract linklists. Threse results are stored as XPath [16] information, pointing into the page. Initial definition procedes in three steps, defining the query interface, master and detail metadata, respectively.

Locating the Query Interface in the Form As a first step, the query interface in the form page is identified by the user by selecting it visually and typing in a query string. As a by-product, the relation between query and resulting master page is defined and enables the wrapper to simulate calls to the form. The management of the query interface is not elaborated further in this work, as it is covered by the components employed.

Caching Master Keys and Detail Links Subsequently, the master result page is screened for terms associated with the query term. WrapIt elects and

visualizes candidates for the records it believes to be the related data. The user can confirm the notion, or repeat the process. In our example, the path `/p/table/a` would retrieve the first movie, 'Shrek 2 (2004)' on a master page. That element contains both key information of the master entry, and the link to the corresponding detail information. Our approach is based on the assumption, that these elements are invariant, while the surrounding regular structure of the page may change. We believe this assumption to hold, because the first represents the underlying business objects, and the second the underlying technical infrastructure. As a result, we can abstract the master into a relation as shown in 1. Following the chain of page links thus creates a cache of all master values associated with a given query. This data forms an implicit first-level cache.

#text	**href**
Shrek 2 (2004)	/title/tt0298148/
Shrek 3 (2006)	/title/tt0413267/
Shrek: Fairy Tale Freakdown (2001)	/title/tt0286967/

Table 1. Sample values of the Detail-Links in the Master-Page

```
<p><b>Partial Matches</b> (7 matches, by popularity)</p>
<p>
 <table> <tr>
   <td valign="top" align="right">1. </td>
   <td valign="top" width="100%">
    <a href="/title/tt0298148/">Shrek 2 (2004)</a></td>
  </tr> <tr>
   <td valign="top" align="right">2. </td>
   <td valign="top" width="100%">
    <a href="/title/tt0413267/">Shrek 3 (2006)</a></td>
  </tr>
  [...]
  <tr>
   <td valign="top" align="right">7. </td>
   <td valign="top" width="100%">
    <a href="/title/tt0286967/">Shrek: Fairy Tale...</a></td>
  </tr></table>
</p>
```

Fig. 2. HTML code excerpt of the MasterPage from `imdb.com`, which results from a query about 'Shrek'.

Caching Detail Field Values Finally, the user selects a detail link. The system repeats this process in the background for other master entries. It now generalizes over the structure of the detail pages looking for varying elements. These are offered to the user as candidates for detail information, as in the preceding step. Also, identification of labels for the detail fields is attempted, as these often

precede the variant information. Thus path information leading to the fields is determined and stored, establishing a schema for detail pages.

```
<b class="ch">Tagline:</b> In summer 2004, they're back for more....
<a href="/rg/title-tease/taglines/title/tt0298148/taglines">(more)</a>
<br><br>
<b class="ch">Plot Outline:</b>
Princess Fiona's parents invite her and Shrek to dinner to celebrate her marriage.
If only they knew the newlyweds were both ogres.
<a href="/rg/title-tease/plotsummary/title/tt0298148/plotsummary">(more)</a>
<a href="/rg/title-tease/trailers/title/tt0298148/trailers">(view trailer)</a>
<br><br>

<b class="ch">User Comments:</b>
Funnier than the original
<a href="#comment">(more)</a>
<br><br>
```

Fig. 3. HTML code excerpt of the DetailPage from `imdb.com`, which results from selecting the link associated with the key 'Shrek 2 (2004)'.

Metadata Schema Figure 4 shows the resulting schema, excluding the data for the query interface. For our example, an instance of MasterPage would refer to a specfic movie query, e.g. with search `term` 'Shrek' and a resulting `URL='http://www.imdb.com/find?tt=on;nm=on;mx=20;q=shrek'`. A MasterItem would be described as `path=p/table/tr/td/a key='Shrek 2 (2004)'` and `location='http://www.imdb.com/title/tt0298148/'`. DetailSchema would lead to an item with the `label` = 'Plot Outline:' and `path=/body/b[@class='ch'][3]`. Finally an associated DetailItem holds the `value = 'Princess Fiona's parents ...'`.

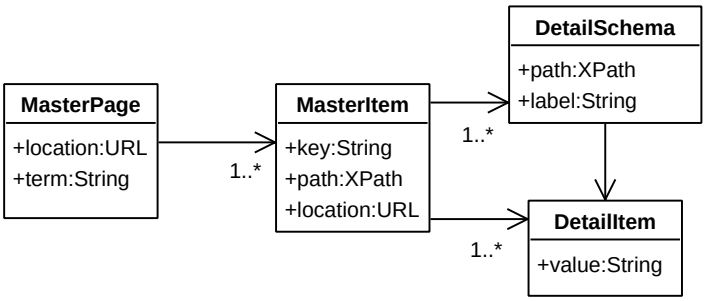

Fig. 4. UML model of the wrapper metadata schema

2.2 Query Submission and Caching

With the configuration introduced above, the wrapper is configured and ready for operation. The user can search terms, and the wrapper will traverse the master/detail structure following the path on the left-hand side of the activity diagram in figure 5, as expected. The branches, that fork of to the right deal with the condition were reconfiguration has to occur, because the cached metadata does not retrieve the expected element, i.e. links to a detail page or the label of an item of the detail. The next section describes how the wrapper deals with such conditions.

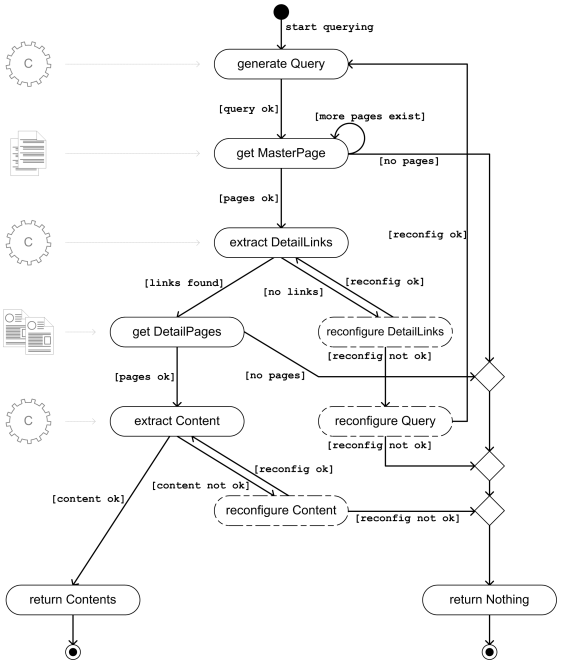

Fig. 5. Activity diagram of the query-process

Query processing has been presented as a linear algorithm. However, since the response to a specific search term is cached in the wrapper, many operations may be executed in parallel. Also, the wrapper can retrieve preliminary answers, while carrying out the query in the background. The wrapper can also choose to skip the query and present the master page using the location stored in the metadata. However none of these options is currently part of our implementation.

2.3 Reconfiguration and Recovery

If the query process as described above fails to find the respective elements in the accustomed places in the page structure, these have probably moved due to aesthetic rearrangement. Thus, the challenge is to realign the structural description. Figure 6 shows a modified master page. The algorithm now selects a MasterPage from its cache, per random, recency, or some other criteria. It queries the source and uses the altered response to locate the MasterItems, which it assumes to be associated with the term. This match can exploit either the key, location or both attributes. If it succeeds, the respective path structure has been rediscovered, and the MasterPage failure circumvented. In our example, `html/body/div[@class="results"]/ul/li/a` would now be the new path value of the MasterItem representing the 'Shrek 2' movie. The details page can be treated similarly. A formalization based on relational algebra of both structures and algorithms has been carried out, to ensure a sound basis of the approach.

```
<div class="results">
 <h1>Search-Results</h1>
 <ul>
  <li><a href="/title/tt0298148/">Shrek 2 (2004)</a></li>
  <li><a href="/title/tt0413267/">Shrek 3 (2006)</a></li>
  <li><a href="/title/tt9876543/">Shrek 2012</a></li>
  [...]
  <li><a href="/title/tt0286967/">Shrek: Fairy Tale...</a></li>
 </ul>
</div>
```

Fig. 6. Modified HTML code of the Master page example

2.4 Architecture

We distinguish three main functions which are covered by the wrapping system *Initial Definition*, *Querying* and *Reconfiguration*. Each of those use cases touches upon different components of the system, as shown in figure7. They are embedded in a framework with a query-interface and a GUI on top of the *Analyzer*, *Extractor* and *Reconfigurator*.

Initial Definition uses the Analyzer for finding structures in pages and the Extractor to present the results to the configurating user. *Querying* is exclusively done by the Extractor. If the extraction fails, a *Reconfiguration* is invoked. This is performed by the Reconfigurator. It uses the Analyzer to find the respective structures and the Extractor to validate its results.

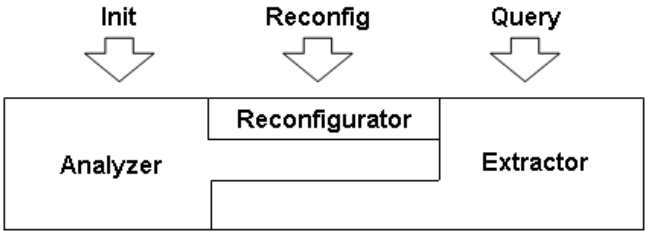

Fig. 7. Architecture

3 Formalization

The following formalization is restricted to the needs of reconfiguration and omits all aspects of value caching. Thus we simplify the model presented in the last section. Without taking into account caching issues we only need to distinguish between master and detail pages. Detail schema information will be stored together with values.

3.1 Relational model

Let $term$ and $location_p$ depict the query term and the URL of the given page. Further M refers to a set of master items, then a cached representation of a user-request to a database-driven website, the master page p, can be depicted in a relational manner as the tuple

$$p = (term, location_p, M).$$

Let further be $location_m$ the URL of one associated detail page, key the text which is enclosed in the corresponding `<A>...</A>` block and $path_m$ the XPath[16] expression which refers to that link. Moreover the set D holds the information which is given at the related detail page, then we define a master item $m_i \in M = \{m_1, m_2, ..., m_n\}$ by the tuple

$$m_i = (key, location_m, path_m, D)$$

Detail schemes and detail items are parts of detail pages and will be formalized together for simplification.
Let be $path_d$ the XPath[16] to a specific detail field, $label$ the describing name of that field, and $value$ a sample instance. Then an element of a detail page $d_j \in D = \{d_1, d_2, ..., d_n\}$ can be defined as a tuple:

$$d_j = (label, value, path_d)$$

3.2 Heuristics

We introduce two heuristics which allow to decide which parts of a certain web page have been structurally altered and help to reassign stored values to the new structure.

Master page A structural change will cause a change of the XPath[16] expression of certain master items. It will not influence the URL nor the label of that item. Thus we define the following heuristics to identify a corresponding master item after a design change has been occurred:

$$\forall m, m' \in M : Location_m(m) = Location_m(m') \wedge Key(m) = Key(m') \Rightarrow m = m'$$

Detail page In contrast to master pages, on detail pages there is no invariant element like a fixed URL. The only common assertion to detail pages in our understanding is a (label,value)-structure, which represents certain properties of the data-object on which the detail page is based. That data structure might not change if structural changes occur, thus we define for two detail elements d, $d' \in D$ the following heuristics to identify the corresponding detail elements after a structural change:

$$\forall d, d' \in D : Label(d) = Label(d') \wedge Value(d) = Value(d') \Rightarrow d = d'$$

3.3 The Reconfiguration process

With help of the above heuristics we employ a reconfiguration function based on content recognition.

Master page Let be S a DOM[17] representation of the (altered) HTML-Structure and M the set of items of a master page which refer to detail pages as stated above. The reconfiguration function f_R extracts for a all tuples $m \in M$ the actualized path expression which refers to certain detail pages:

$$\begin{aligned} f_R(M, S) := \{m' | m' \in M \wedge Location_m(m') = Location_m(m) \\ \wedge Key(m') = Key(m) \wedge Path(m') = Path(S.m)\} \end{aligned}$$

The reconfiguration process is done iteratively for all changed elements. Figure 8 shows the algorithm in detail.

Detail page The application of the reconfiguration for a detail page works similarly to that one for the master page as stated above. The only difference is the used heuristics.

```
Reconfig(M,S)
for all node in S {
   if node = ('<a>...<a>') then
    for all m in M {
       if (href(node) = Location(m) and subnode(node).text = Label(m))
          then Path(m) = node.getPath();
       fi;
     }
  fi;
}
```

Fig. 8. The Reconfiguration algorithm for master pages.

4 Related Work

Our approach shows similarities to guided configuration used in W4F [14] and XWrap [10]. Analysis mainly uses two technologies: Heuristics, analyzing the DOM [17] tree of a document in order to find groups of links that point to Detail-Pages, are inspired by XWrap [10] and IEPAD [2], content-lookup in Detail-Pages is based on the Roadrunner [5] technique of solving mismatches across documents, similarly found in WrapIt [12]. Extraction also uses Roadrunner and WrapIt, based on comparison of a page with a general wrapping representation. Furthermore, extraction employs XPath [16] queries to retrieve Detail-Links. A specialty of our system is the ability of reconfiguring itself at runtime without user intervention. This technique is inspired by the Content-Based-Recognition introduced in WrapIt and is supported by similar analysis-techniques as used during the configuration. Additional modules, e.g. for label-assignment and form-filling like in DeLa [18] or HiWE [13], will be integrated in the future.

5 Future Work

The presented process of reconfiguration with Content Based Recognition is a good way for designing robust wrapper systems. Nevertheless there will be still cases where our system could not be reconfigured satisfactorily. Furthermore our system is designed only for the described schema of web sources yet and we have to generalize it to work also with other schemes of web sources.

5.1 Fortifying the approach

At the time the approach lacks in terms of robustness. In case if cached sample values are no more present in the underlying database of the website at the time of reconfiguration, the process will fail. High frequency changing data sources, e.g. flight schedules or movie sites with performance calenders, are not suitable

for the approach. Generally a design change could not be reconfigured with our approach even if the data-structure remains identical if none of the stored sample-values can be retrieved at a later date. To solve this problem we are working on matching techniques that do not rely on certain sample values but rather search and compare actual values on and between Master- and Detail-Pages.

Other problems concern overlap of page structure and sample values, leading to deduction of misleading path entries. An example are repeated occasions of identical values on a page, e.g. the headline of the page is also the title of the book that is shown there. Another problem are attributes that have similar values, e.g. date attributes often just contain year dates that induce a high overlap. Thus the reconfiguration results, especially the new path entries, need to be validated over several pages to detect erroneous configuration data.

5.2 Generalizing the approach

The approach can be adapted to other schemes of web sources, due to the main idea of storing sample values independent from the structure information. If you have the new structure of a page that contains the stored data and compare it with the stored data you can draw conclusions from it to the new positions of the defined attributes in the changed structure. In general the approach is independent from the described Master-/Detail-Page presentation. The main problem is to find a way to get one or more instances of the stored data from the changed website.

References

1. N. Ashish and C. Knoblock. Wrapper generation for semi-structured internet sources. In *Proc. Workshop on Management of Semistructured Data*, Tucson, 1997.
2. Chia-Hui Chang. IEPAD: Information extraction based on pattern discovery. In *Tenth International World Wide Web Conference*, pages 681–687, 2001.
3. W. Cohen, M. Hurst, and L. Jensen. A flexible learning system for wrapping tables and lists in html documents, 2002.
4. W. Cohen and L. Jensen. A structured wrapper induction system for extracting information from semi-structured documents.
5. Valter Crescenzi, Giansalvatore Mecca, and Paolo Merialdo. Roadrunner: Towards automatic data extraction from large web sites. In *The VLDB Journal*, pages 109–118, 2001.
6. Saikat Mukherjee Guizhen. Automatic discovery of semantic structures in html documents, 2000.
7. Thomas Kabisch. Grammatikbasiertes semantisches wrapping fuer föderierte informationssysteme. In *Tagungsband zum 15. GI-Workshop Grundlagen von Datenbanken*, pages 62–66. Fakultät fuer Informatik,Otto-von-Guericke-Universität Magdeburg, 2003.
8. A. Laender, B. Ribeiro-Neto, A. Silva, and J. Teixeira. A brief survey of web data extraction tools. In *SIGMOD Record*, volume 31, June 2002.

9. Ling Liu, Wei Han, David Buttler, Calton Pu, and Wei Tang. An XML-based wrapper generator for web information extraction. In *SIGMOD Conference*, pages 540–543, 1999.
10. Ling Liu, Calton Pu, and Wei Han. XWRAP: An XML-enabled wrapper construction system for web information sources. In *ICDE*, pages 611–621, 2000.
11. I. Muslea, S. Minton, and C. Knoblock. Stalker: Learning extraction rules for semistructured, 1998.
12. Mattis Neiling, Markus Schaal, and Martin Schumann. WrapIt: Automated integration of web databases with extensional overlaps, 2002.
13. Sriram Raghavan and Hector Garcia-Molina. Crawling the hidden web. In *Proceedings of the Twenty-seventh International Conference on Very Large Databases*, 2001.
14. Arnaud Sahuguet and Fabien Azavant. Building intelligent web applications using lightweight wrappers. *Data Knowledge Engineering*, 36(3):283–316, 2001.
15. Myra Spiliopoulou and Lukas C. Faulstich. WUM: a Web Utilization Miner. In *Workshop on the Web and Data Bases (WebDB98)*, pages 109–115, 1998.
16. W3C. XML path language (XPath) version 1.0 – W3C recommendation 16 november 1999. Available at http://www.w3.org/TR/xpath.html, 1999.
17. W3C. Document object model (DOM) level 3 core specification version 1.0 – W3C recommendation 07 april 2004. Available at http://www.w3.org/TR/DOM-Level-3-Core, 2004.
18. Jiying Wang and Fred H. Lochovsky. Data extraction and label assignment for web databases. In *Twelft International World Wide Web Conference*, pages 470–480, 2003.